开源魅力

面向Web开源技术整合开发与实战应用

马洪江 周相兵 佘堃 等编著

- ✓ 结构完善、体系清晰
- ✓ 知识面广、内容翔实
- ✓ 原理、技术与实战完善结合
- ✓ 经验、体验整合的结晶

清华大学出版社

开源魅力：面向 Web 开源技术整合开发与实战应用

周相兵　马洪江　佘　堃　编著

清华大学出版社
北　京

内 容 简 介

本书不仅是一本 J2EE 入门图书，还详细地介绍了面向开源软件的构架原理、分析设计方法、开发方法、开发技术和众多当前流行的开源框架。重点分析介绍了 SSH(i)（Spring、Struts、Hibernate（和 iBatis）、A2J（Axis/CXF、WSDL2OWL、Jena）和页面处理（AJAX、Direct Web Remoting、Portlet（Jetspeed、Liferay）等开源软件集成方法、基本应用方法和案例。书中也详细分析介绍了面向开源软件的软件开发模式，即用软件工程知识和软件开发方法将基于 Web 的开源软件集成在一起，并使这些开源软件松散耦合地组织在了一起。书中最后配备了两个实用性强的案例来进一步分析面向开源软件的软件开发方法。这两个案例都是以本书中介绍的开源软件为基础，以便读者更快速地掌握 Java EE（J2EE）应用开发技术。本书配套的光盘内容包括了两个案例的源代码，也可作为读者掌握这一门软件开发方法的指导具体案例。

图书在版编目（CIP）数据

开源魅力：面向 Web 开源技术整合开发与实战应用 / 周相兵，马洪江，佘堃编著. —北京：清华大学出版社，2013.3

ISBN 978-7-302-30926-0

Ⅰ. ①开…　Ⅱ. ①周…　②马…　③佘…　Ⅲ. ①软件开发　Ⅳ. ①TP311.52

中国版本图书馆 CIP 数据核字（2012）第 286736 号

责任编辑： 夏兆彦
封面设计：
责任校对： 徐俊伟
责任印制：

出版发行： 清华大学出版社
网　　址：http://www.tup.com.cn, http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　　销： 全国新华书店
开　　本： 190mm×260mm　　印　张：44.5　　字　数：1111 千字
版　　次： 2013 年 3 月第 1 版　　印　次：2013 年 3 月第 1 次印刷
印　　数： 1~
定　　价： 　　元

产品编号：041047-01

前　　言

近年来面向 Web 开发的开源件已在各行各业的软件领域得到了广泛的应用，这就使得面向开源软件的软件开发逐渐形成了一种新的软件开发方法。但怎样使其成为一种软件开发体系和方法是目前摆在众多软件研发人员面前的难题，因此编者根据多年的 Java EE（J2EE）开发经验、对开源软件的深入研究成果，以及国内开源软件的相关文献和 developerWorks 知识来解决这个问题：本书全面介绍面向 Web 的开源件原理方法、技术构成、集成方法、开源件技术、应用体系和面向服务的软件技术；同时，将现阶段流行的、基于 Web 的开源件整合成一个中间件；并以该中间件为基础，举例了 Web 服务和语义 Web 服务的开发方法，并开发了一个科研统计分析系统。全书在撰写过程特别注重可读性、实用性、专业性、学术性、可操作性，并力求知识面广、难易合理搭配，从而让读者不用翻阅太多的资料就可以从事相关软件研究与开发工作。因此，本书首先从原理、方法、技术层面进行全面分析研究，然后从具体的开源件进行分析研究，最后总结面向开源件框架的轻量级软件开发方法和流程。

全书内容共分 9 章，分别如下：

第 1 章　开源软件发展的概况

第 2 章　面向开源软件的软件构架原理

第 3 章　面向开源软件的分析设计方法

第 4 章　面向开源软件的软件开发方法

第 5 章　面向开源软件的软件开发技术

第 6 章　面向开源软件的软件开发开源框架

第 7 章　多开源软件框架整合方法

第 8 章　SAJP-M 轻量级开源中间件整合实现

第 9 章　用 SAJP-M 设计实现科研绩效系统

本书的特色主要表现在：

1. 开源软件方法集成、开源软件技术集成

开源软件方法集成就是根据开源软件的发展概况，以软件工程思想知识来组织当前流行的、基于 Web 的开源软件，从而形成一种面向开源软件的软件开发方法。开源软件技术集成就是通过基于 XML 的配置文件来集成当前流行的、基于 Web 的开源软件，使其成为一种面向开源软件的软件开发中间件模式。

2. 经验丰富、针对性强、技术价值高、技术总结区分比较

编者有着丰富的 Java EE（J2EE）软件开发经验，又从事软件开发相关课程教学，还有着长期从事开源软件技术研究工作的经历。因此，编者明白读者需要什么样的书籍，即需要理论联系技术，技术联系应用的体系化的书籍，也需要书中对同一类知识、技术进行对比分析。

3．内容翔实、实用性强、层次逻辑分明、难易结合

本书介绍了面向开源软件的发展历程，详细分析了面向开源软件的软件构架原理、方法和技术，以及具体的应用案例；针对不同的开源软件列出了具体的应用方法和集成原理。严格按照 Java EE（J2EE）及相关规范进行本书的知识布局，使得只要读者参考本书内容，就完全可以身临其境地感受企业实际的开发。

本书第 1、2 章由马洪江、周相兵编写，第 3 章由周相兵编写，第 4 章由马洪江、余堃编写，第 5 章由周相兵、马洪江编写，第 6、7、8、9 章由马洪江、周相兵、余堃共同完成编写，全书由马洪江统稿。

本书既可以作为 Java EE（J2EE）初学者的书籍，也可以作为 Java EE（J2EE）应用开发者的提高指导，还可以作为高年级的实践指导用书。

限于编者的水平和时间，在本书中难免存在错误，希望读者给予指导并与我们进行交流，以便本书的修订，使之更能符合读者的要求。

本书在成书过程中，得到了 IBM 软件设计师谢成锦、徐海的大力支持，也得到了成都理工大学苗放教授和电子科技大学国家级计算机实验教学示范中心有关人员的支持，在这里一并感谢他们。

最后，我们衷心感谢清华大学出版社的编辑为此书出版付出的努力和辛勤劳动。

编　者

2012 年 1 月

目　　录

第0章 概 论

自从WWW（World Wide Web，简称为Web）在20世纪90年代初正式由万维网联盟[World Wide Web Consortium，W3C，其发明者是美国Timothy John Berners-Lee（简写为Berners-Lee，现任W3C主席），他早在20世纪80年代初就给出了建议和原型。]发布以来，使各行各业的信息化建设和发展逐渐发生了本质性的变化，这主要表现在：

① 使信息化建设真正走向网络化发展；

② 通过Web浏览器给用户操作带来了便捷性；

③ 信息发布与获取更加简单、容易；

④ 改变了传统的信息化建设模式，加快了信息化建设速度；

⑤ 促进了网络技术快速、飞猛的发展，特别是软件技术的发展，使得了软件技术向多元化、多样化方向发展；

⑥ 催生了基于Web的新生网络应用，如论坛（BBS）、社交网络（SNS）、博客（BLOG）、微博（Micro Blog）、电子商务及与电子相关的新兴产业等；

⑦ 促使信息化系统的网络结构发生了转变，由最初的客户/服务（Client/Server，C/S）向单一的浏览器/服务器（Browser/Server，B/S）模式转变，以及向多层次、多结构（如点对点（Peer-to-Peer，P2P）结构等）的B/S的分布式系统转变，改变了对传统网络信息系统的认识；

⑧ 促进对信息化的认识从最初的以业务为中心的到当前以需求为中心方向转变；

⑨ 促使了信息化的网络计算模式发生了变革，由最初基本网格计算模式向并行计算、服务计算、网格计算、云计算和绿色计算方向发展，以及向多级别、多结构、多集群、多端、多层次、多需求等综合性的分布式计算；

⑩ 使得网络信息化应用向自由和共享角度发展，特别对软件技术影响起到了非常重要的作用，最明显的就是催生开源软件（Open Source Software，OSS）的发展与进步。

随着Web技术的发展，促进面向Web的开源软件的快速发展，也使得开源软件在软件研发过程中占有重要的地位；使得出现了一大批可用度高、质量好、健壮性稳定的开源软件，以及在各行各业信息化建设中起到关键作用的大量优秀的各类型的开源软件，最具有代表的就是以Sun公司（现已被ORACLE收购）的Java为基础语言的开源软件阵营，她目前在信息化建设领域占很大的比重，最为成熟就是J2EE（Java 2 Platform Enterprise Edition）为框架的结构，大多数的开源软件都是以J2EE中的JDK（Java Development Kit）为基础进行扩展、深化和开发的；又特别是大多数基于Web技术开源软件的都是以J2EE为基础的，包括正在发展的面向服务和面向智能分析设计的信息化系统，其中许多相关开源软件已陆续问世。

开源软件的发展离不开W3C、结构化资讯标准促进组织（Organization for the Advancement of Structured Information Standards，ORSIS）组织所提供的相关标准和技术的支持，这两个非营利组织是目前基于Web的开源软件的重要支撑，它为开源软件的应用、兼容和扩展提供了统一的、规范的说明和实践方法；同时，开源软件的持续和健康发展也离不开

这两个组织继续修改、更新和推出适应 Web 技术发展的标准和规范。

1．WWW 发展概况

而 WWW（Web）的核心部分由统一资源标识符（Uniform Resource Identifier，URI）、超文本传送协议（HyperText Transfer Protocol，HTTP）、超文本置标语言（Hypertext Markup Language，THML；习称超文本标记语言）这三个基本标准组成。

1）URI、URL、URN

URI 是一个用于标识某一互联网资源名称的字符串，并允许用户对网络中的资源通过协议进行交互操作，由存放资源的主机名、片段标志符和相结 URI 组成。即 URL 是标识一个互联网资源，并指定对其进行操作或取得该资源的方法的 URI，

URL 是因特网上标准的资源的地址，可以认为 URL 是 URI 的一个变种，或是命名机制的一个子集。URI 是确定一个因特网资源，而 URL 不但确定一个资源，还表示这个资源在哪里，因此，采用 URL 可以用一种统一的格式来描述文件、服务器的地址和目录等资源。URL 由协议、主机 IP 地址和主机资源的具体地址三部分组成。

URN 即统一资源名称（Uniform Resource Name，URN）如同一个人的名称，URL 就代表一个人所在的地址，URN 则定义某事物的身份，不依赖资源所处的位置，且有可能减少失效连接的个数；而 URL 提供查找该事物的方法。因此，URL 和 URN 有着互补的作用。

2）HTTP、HTTPS

HTTPS 即超文本传输协议安全（Hypertext Transfer Protocol over Secure Socket Layer，HTTPS）是 HTTP 和安全套接层协议层（Secure Socket Layer，SSL）/传输层安全（Transport Layer Security，TLS）的组合，用以提供通信加密及对网络服务器身份的鉴定，它常用于因特网（Internet）上的交易支付和企业信息系统中敏感信息通信。即在 HTTP 层下加 SSL/TLS 层，因此，HTTPS 的安全基础是 SSL，但对上层协议一无所知，所以 SSL 服务器只能为一个 IP 地址/端口组合提供一个证书[1]。HTTPS 和 SSL 支持使用 X.509（是公钥证书、证书吊销清单、属性证书和证书路径验证算法等证书标准）数字认证，也可以认为 HTTPS 是建立一种安全信息通道，以及另一种确认网站的真实性。而 HTTP 与 HTTPS 的区别主要有：

① HTTP 的 URL 表示为"http:// "，使用的默认端口是 80；HTTPS 的 URL 表示为"https://"，使用的默认端口是 443。

② HTTP 是明文传输，因此不安全，攻击者容易通过“监听”和“中间人攻击”等手段获取网站账户和敏感信息等。HTTPS 具有安全性的 SSL 加密传输协议，被设计为可防止 HTTP 所带来的攻击，并认为是安全可靠的；但 HTTPS 并不能防止站点被网络蜘蛛抓取；在某些情形中，被加密资源的 URL 可仅通过截获请求和响应的大小推得[2]。

③ HTTPS 中的 TLS 有简单和交互两种策略，交互策略更为安全，但需要用户在他们的浏览器中安装一个 CA（Certificate Authority）来进行认证，而且需要交费。

④ HTTP 属于简单连接、无状态的；但 HTTPS 协议是由 SSL/TLS+HTTP 协议构建的可进行加密传输、身份认证的网络协议，因此 HTTPS 比 HTTP 协议更安全。

---

① Apache FAQ: Why can't I use SSL with name-based/non-IP-based virtual hosts?.http://httpd.apache.org/docs/2.0/ssl/ssl_faq.html#vhosts

② Pusep, Stanislaw. The Pirate Bay un-SSL. 2008-7-31.http://sysd.org/stas/node/220

但 HTTPS 与安全超文本传输协议（Secure Hypertext Transfer Protocol，S-HTTP）是有区别的，S-HTTP 是一种面向安全信息通信的协议， S-HTTP 定义于 RFC 2260[①]中。在语法上，S-HTTP 报文与 HTTP 相同，可以与 HTTP 结合使用，能与 HTTP 信息模型共存并易于与 HTTP 应用程序相整合，是一个 HTTPS URI Scheme 的可选方案。因此，在市场上，HTTPS 远胜于 S-HTTP。

3）HTML、SHTML

当前使用的 SHTML（Server Side Include HyperText Markup Language）是通过 SSI（Server Side Include）扩展 HTML 文件名，通常称为服务端嵌入，是为 Web 服务提供一套可以直接嵌入到 HTML 中的命令，当浏览器端访问这些 SHTML 文件时，服务器端会把这些 SHTML 文件进行读取和解释，并把 SHTML 文件中包含的 SSI 指令解释出来，因此也可以认为 SHMTL 是服务器动态产成的 HTML，它的程序格式为<!--#指令名称="指令参数">，也可以认为是一种类似于 ASP（Active Server Page）的基于服务器的网页制作技术（相信熟悉 ASP 的读者，就非常容易理解和掌握 SHMTL 的应用方法）。通常情况下，使用 SHTML 时，需要明确如下 SSI 基本指令格式和意义：

① 将浏览器能识别的文档内容直接插入到动态文档中<#include>，例：<!--#include file="test.html"-->。

② 显示服务器端环境变量<#echo>，例：<!--#echo var="REMOTE_ADDR"-->，表示显示获取浏览器用户的 IP 地址。

③ 直接在服务器上执行各种程序<#exec>，例：<!--#exec cgi="/cgi-bin/test.cgi"-->，表示将执行 CGI 程序 test；<!--#exec cmd="dir /b"-->，表示将会显示当前目录下文件列表。其中"#exec cgi="用来专门执行 CGI 程序，"#exec cmd="用来执行 shell 命令。

④ 设置 SSI 信息显示格式<#config>，高级 SSI<XSSI>可设置变量和使用 if 条件语句。其中 XSSI（Extended SSI）是一组高级 SSI 指令，内置于 Apache 1.2 或更高版本的 mod-include 模块中，一般可以使用的指令有#printenv、#set、#if；其中：

#printenv 表示显示当前存在于 Web 服务器环境中的所有环境变量，例：直接使用<!--#printenv-->就可以了。

#set 表示给变量赋值，以用于 if 语句，例：<!--#set var="test"value="测试"-->。

#if 表示创建可以改变数据的页面，这些数据根据使用 if 语句时计算的要求予以显示（最重要的目的之一就是用来生成动态的 HTML）。例：

```
<!--#if expr="$test=\"this is a test\""-->
这是一个测试；
<!--#elif expr="$test=\"this is not a test\"" -->
这不是一个测试；
<!--#else-->
是一个测试吗？
<!--#endif"-->
```

① E. Rescorla,A. Schiffman.The Secure HyperText Transfer Protocol(RFC 2660),http://tools.ietf.org/html/rfc2660

⑤ <#flastmod file>显示 Web 文档的 file 所指文件最近更新日期，表示把当前目录下的网页页面插入到当前页面，例：<!--#flastmod file="test.html"-->，其中 file 表示指定页面相对于 test.html 的位置。

⑥ <##fsize file>显示 Web 文档的 file 所指文件的长度，表示将当前目录下网页页面的文件大小入到当前页面，例：<!--#flastmod file="test.html"-->。

Web 服务器在处理网页的同时处理 SSI 指令，当 Web 服务器遇到 SSI 指令时，直接将包含文件的内容插入 HTML 网页。当在使用 SSI 将内容发送到浏览器之前，可以使用 SSI 指令将文本、图形或应用程序信息包含到网页中，所以对 Web 网站结构固定且需要维护时，把一些有限的 HTML 模块放在 Web 服务器上，然后把需要更新的网站信息传到服务器就可以方便、简单、轻松地实现网页更新，自动生成网页，这样会有效提高大型网站管理效率。

而 SHTML 与 HTML 的区别有以下两点：

① SHTML 是一种 Web 服务器 API（Application Program Interface），而 HMTL 不是。

② SHTML 是 Web 服务器动态生成的 HTML，是一种用 SSI 技术的超文本书件格式；而 HTML 是一种静态页面格式，没有在服务端执行脚本。

4）THML、XML

可扩展置标语言（Extensible Markup Language，XML）是一种跨平台的、依赖于内容、处理半结构化的一种处理置标语言，与 HMTL 同属于标准通用置标语言（Standard Generalized Markup Language，SGML），被认为 XML 是对 HTML 一种扩展，这是由于：

① HTML 不能解决所有解释数据的问题、难以扩充、不具备弹性、易读性不好、处理效能不佳。

② XML 通常用来传送及携带数据信息，不用来表现或展示数据，HTML 语言则用来表现数据，XML 也有严格的语法结构。

③ XML 扩展性比 HTML 强，XML 的语法比 HTML 严格，XML 与 HTML 可以互补。

同时，XML 在传送、携带数据信息时，具备严格的语法判定、认识能力，而且具备处理这些数据信息的能力，如数据索引、排序、查找、相关一致性检查等，这些工具包括 XPath、Xquery 等，而 XML 编辑器主要有 XML Notepad、XML Spy、Xeena 等。XML 的文档对象模型（Document Object Model，DOM）将 XML 文档作为一个树形结构，而树叶被定义为节点来组织这些信息，因此具有良好的层次结构和树结构的特性，并通过文档类型定义（Document Type Definition，DTD）和 Schema（XML 模式）进行解析；利用可扩展样式表转换语言（EXtensible Stylesheet Language Transformation，XSLT）来操纵 XML 数据，他将 XML 文档转换为不同 XML 结构的文档，甚至还可以转换为非 XML 文档。因此，XML 相比 HTML 具备以下优势：

① XML 具备跨平台、跨语言、跨结构的能力，即可以搜索、查找不同平台、不同结构、不同语言的数据信息。

② XML 可以用于存储数据，使结构化的数据表示为半结构化、非结构化数据，这样便于不同的数据源或不同请求访问。

③ XML 可以用于共享数据，由于 XML 是以纯文本的形式存储数据，因此与软件、硬件的状态无关，使得不同的需求源可以使用 XML。

④ XML 可以用于交换、利用数据，由于 XML 自由的文本格式，可以作为数据交换的中

间过渡介质，使不同的数据格式间相互转换，从而提高数据的使用能力。

⑤ XML 用于多语言、多平台融合和部署配置，利用 XML 可以方便的将要使用的多种开发语言，支持平台融合在一起进行使用，并能将所开发的应用程序进行部署。

⑥ XML 可以从 HTML 中分离数据，确保数据改动时不会导致 HTML 文件也需要改动，因此大大降低了网页维护难度。

⑦ 通过 XML 可以与电子商务、各类企业系统、各类商业系统、各类信息化集成系统、电子政务等进行数据信息通信，从而提高各系统使用效率。

⑧XML 可以作为创建其他新语言，如 XML 就是 WAP 和 WML 语言的基础。

2. Web 技术概况

Web 技术是实现网页页面的技术，当然也包括前面所述的 HTTP、HTTPS、SHTML、XML 等技术。Web 是一种典型的分布式应用结构，因此每一次数据信息交换都要涉及客户端和服务端。当前，实现 Web 的技术包含了许多种类型，如 HTTP、XHTML、CSS、JavaScript、DOM、Java servlet、XML 及相关技术、JSP、SOAP、Web 服务、JAX-RPC、CGI、ASP、SQL、ASPX、PHP 和 ColdFusion 技术等。其中：

① 有包括 W3C 标准的 HTML、XHTML、CSS、XML 等；

② 客户端的 HTML 语言、XML 语言、Java Applets、脚本程序、CSS、DHTML、XHTML、插件技术以及 VRML 技术等；

③ 服务端的 CGI、PHP、ASP、ASP.NET、Servlet 和 JSP 技术等。

3. 语义 Web 技术概况

语义 Web（Semantic Web）在 2001 年由 Berners-Lee、Hendler 和 Lassila 三人正式提出（早在 1998 年，Berners-Lee 就提出了语义 Web 的概念），它是对 Web 的扩展，是计算机业和互联网业对网络下一阶段发展所作出的术语化定义；它的核心是通过给 WWW 上的文档添加能够被计算机所理解的语义，从而使整个互联网成为一个通用的信息交换媒介；它的关键技术是资源描述框架（Resource Description Framework，RDF）、XML、本体（Ontology），如图 0 所示。

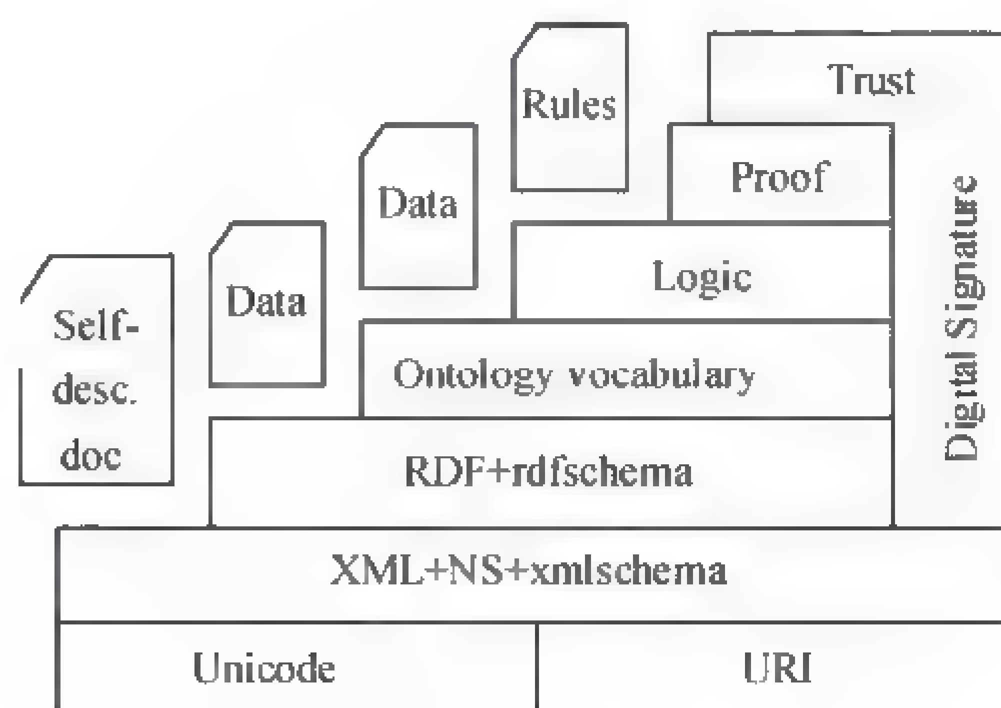

图 0　语义 Web 结构图（In W3C）

与此同时，Berners-Lee 在 2000 年也正式提出了七层语义 Web 层次结构（又称语义 Web 堆栈），并指出语义 Web 是建立在 XML 之上的，从而使 Web 成为语法与语义两重标准的智能化网成为可能，并能使计算机能判断和识别数据信息。其主要思想是 Web 与语义，Web 主

要包括资源静态映射的 URI、创立可导航的空间、通信交互的共享空间和自描述文档，语义主要包括机器可以认识、希望得到什么信息、信息特征间怎样转换和表达方式。在语义 Web 技术上主要体现在以下几个方面：

① 当前多种 WWW 上的技术都能适应用语义 Web 中，这为语义 Web 发展奠定了基础；

② 在语义 Web 中，需要得到基础的国际资源标识符（Internationalized Resource Identifier，IRI）、Unicode、XML 和 XML 命名空间等的支持。其中，IRI 作为 URI 的泛化形式，提供对语义网资源加以唯一标识的手段；Unicode 表示采用多种语言来表现和处理文字，XML 命名空间是作为语义 Web 连接的访问方式。

③ 当前支持语义 Web 的技术包括 RDF、RDF Schema（RDFS）、网络本体语言（Web Ontology Language，OWL）、SPARQL、规则交换格式（Rule Interchange Format，RIF）等。其中，RDFS 是 RDF 的基础词汇表；OWL 是对 RDF 的扩展，用来描述、声明 RDF 的语义构造，是由描述逻辑（Description Logic）为基础的，因此，具备传递性、判定性等特征，且具有推理的能力，为丰富语义 Web 提供前所未有的帮助；SPARQL 是一种 RDF 查询语言，用于查询任何基于 RDF 的数据。

④ 而对于语义 Web 的用户界面、信任、验证和加密等，还处于发展中。

综上，对面向 Web 的一些基础知识进行了概述、比较，为面向 Web 开源技术整合与实践应用提供一定的基本知识一览，为快速进行本书学习提供预备知识。

第 1 章　开源软件发展的概况

当前，开源软件的应用已经扩展到诸多行业，对推动软件业迅速发展起到了不可估量的作用，这也体现了软件的自由和共享宗旨；软件开发者只需要根据开源软件的开发社区可以方便获得应有的疑惑、问题，方便改善自己的编程风格，以及根据版权持有人在软件协议的规定之下保留一部分权利并允许用户学习、修改、增进提高这款软件的质量、软件的健壮性和可用性；当然对于开源软件所组成的源代码开放，并不一定是开源软件，它只是一种共享形式，一种可以任意获取的计算机软件。

1.1　开源软件的定义

在介绍开源软件定义之前，需要回顾一下开源软件的基础自由软件，也就是说，开源软件的产生是由自由软件衍生而来。同时，也简要介绍中国开源软件推进联盟。图 1-1 所示是自由软件与非自由软件的分类结构图。

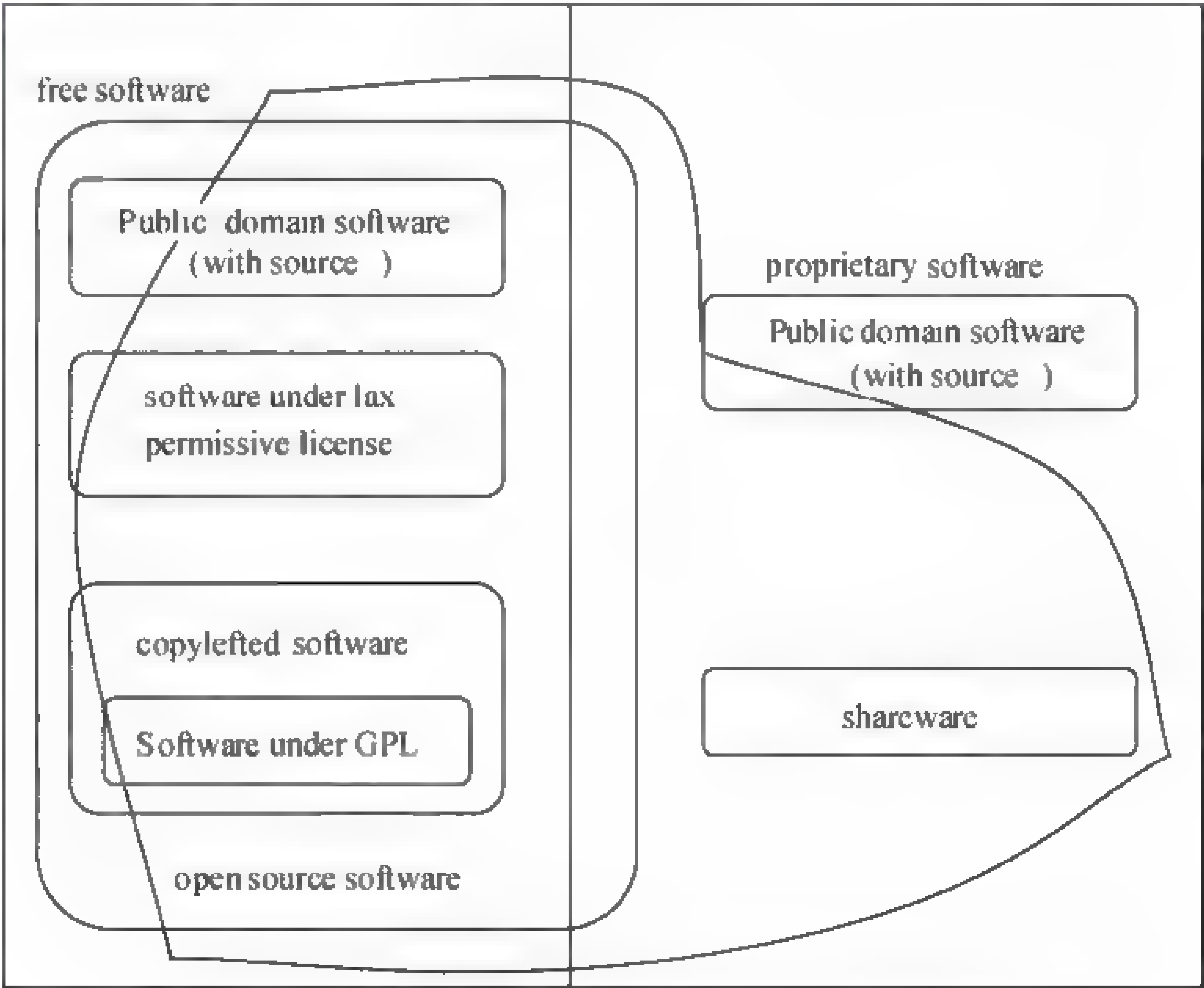

图 1-1　自由软件与非自由软件的分类结构图
（来源：http://www.gnu.org/philosophy/categories.html）

1.1.1　自由软件定义

自由软件（Free Software）是由自由软件基金会（Free Software Foundation，FSF）定义

为一种可以不受限制地自由使用、复制、研究、修改和分发的软件。而 FSF 是一个致力于推广自由软件的美国民间非营利性组织，是由 Richard Matthew Stallman 在 1985 年 10 月创建，主要工作是执行 GNU(GNU's Not Unix)计划，以及开发更多的自由软件。GNU 计划是 Richard Matthew Stallman 在 1983 年 9 月 27 日公开发起的，其目标是创建一套完全自由的操作系统。为保证 GNU 软件可以自由地"使用、复制、修改和发布"，所有 GNU 软件都包含一份在禁止其他人添加任何限制的情况下，授权所有权利给任何人的协议条款，GNU 通用公共许可证（GNU General Public License，GPL）[①]。到目前为止，GPL 经过了三个版本，最近一个版本是在 2007 年 6 月 29 日由自由软件基金会正式发布的 GPL 第 3 版[②]。

GPL 是一个广泛使用的自由软件许可协议，这个就是被称为"公共版权"的概念。同时，GNU 也针对不同场合，提供 GNU 宽通用公共许可证（与 GNU 自由文档许可证这两种协议条款。

在 GNU 工程中，GPL 给予了计算机程序自由软件的定义，并且使用 Copyleft 来确保程序的自由被完善的保留，即通常使用 copyleft 这类许可方式来保护每个使用者都享有这些软件自由，但是非 copyleft 的自由软件也同时存在。而 Copyleft 是由自由软件运动所发展的概念，是一种利用现有著作权体制来挑战该体制的授权方式，允许他人任意的修改和散布作品，在自由软件授权方式中增加 copyleft 条款之后，该自由软件除了允许使用者自由使用、散布、改作之外，copyleft 条款更要求使用者改作后的衍生作品必须要以同等的授权方式发布。当然，Copyleft 授权方式虽然与常见的著作权授权模式不同，但选择 copyleft 授权方式并不代表作者放弃著作权，反而更是尊重自由软件的全自由和共享，强制被授权者使用同样授权发布衍生作品，copyleft 许可协议不反对著作权的基本体制，却是透过利用著作权法来进一步地促进创作自由。

除了 GPL 协议，通常还使用 BSD（Berkeley Software Distribution license）许可证，是自由软件中使用最广泛的许可证之一，即 BSD 软件就是遵照这个许可证来发布。与 GNU 通用公共许可证（GPL）到限制重重的著作权（Copyright）相比，BSD 许可证比较宽松，甚至跟公有领域更为接近。事实上，BSD 许可证被认为是 copycenter（中间版权），界乎标准的 copyright 与 GPL 的 copyleft 之间[③]，而 copyright 的概念是借由赋予对著作的专有权利的方式提供作者从事创作之经济动机，但相对的此种赋予作者专有权利的方式同时也限制了他人任意使用创作物的自由。GPL 与 BSD 许可协议相比，主要区别就在于 GPL 寻求确保自由能在复制件及演绎作品中得到保障，即通过一种叫 Copyleft 的法律机制实现，即要求 GPL 程序的演绎作品也要在 GPL 之下。而 BSD 式的许可协议并不禁止演绎作品变成专有软件。

自由软件所指称的软件，其使用者有使用、复制、散布、研究、改写、再利用该软件的自由。更精确地说，自由软件赋予使用者四种自由[④]：

（1）不论目的如何，有使用该软件的自由。

（2）有研究该软件如何运作的自由，并且可以不改写该软件来符合使用者自身的需求。并获取该软件的源码来满足不同需求。

---

① http://www.gnu.org/licenses/gpl.html

② http://www.gnu.org/licenses/gpl-3.0.html

③ http://catb.org/~esr/jargon/html/C/copycenter.html

④ http://www.gnu.org/philosophy/free-sw.zh-cn.html

（3）有重新散布该软件的自由，每个人都可以藉由散布自由软件来敦亲睦邻。

（4）有改善再利用该软件的自由，并且可以发表改写版供公众使用，如此一来，整个社群都可以受惠。如前项，取得该软件之源码为达成此目的之前提。

如果一软件的用户具有上述四种权利，则该软件得以被称之为“自由软件”。也就是说，用户必须能够自由地、以不收费或是收取合理的散布费用的方式、在任何时间发布该软件的原版或是改写版，在任何地方给任何人使用。如果用户不必问任何人或是支付任何的许可费用从事这些行为，就表示他拥有自由软件所赋予的自由权利。但是这不表明：

（1）自由软件并不是没有版权。部分的自由软件可以免费取得，并且它的源代码可以自由修改并散布但它并不是没有版权。版权是当某项作品完成时就自然产生了，不需申请或注册。

（2）自由软件并不使用封闭格式。封闭软件通常会使用专属的封闭格式，但这会极大地限制了用户的自由度。而自由软件则完全不同，由于自由软件全自由和共享的，所以它所使用的任何格式都是透明的。因此，自由软件永远不会利用专属的封闭格式来限制用户分发或修改的自由。

（3）GPL 并不是没有兼容性。如 MIT/X 许可协议、BSD 许可协议、LGPL 等大多数自由软件许可协议是与 GPL 兼容的，即它们的代码与 GPL 代码混用无冲突。

（4）自由软件并不是没有版权限定。GPL 文本是有版权的，且著作权人是自由软件基金会。但是，自由软件基金会没有在 GPL 下发行作品的著作权（除非作者指定自由软件基金会是著作权人）。因此通常认为，只有著作权人才有权对许可协议的违反进行起诉，但是那并不准确[①]。同时，由软件基金会允许人们使用以 GPL 为基础的其他许可协议，但不允许演绎的许可协议未经授权地使用 GPL 的前言，不过像这样的许可协议通常与 GPL 不兼容[②]。

虽然自由软件从理想角度可以促进软件进步、发展、自由和共享，推动软件业“全民化”，但其“自由度”过大，共享程度过于宽松，并且 GPL 非常强调软件代码和使用上的开放性、透明性，一旦一个软件使用了 GPL 的软件代码，GPL 就要求这个软件必须使用 GPL 作为其许可证，影响了部分商业软件公司的利益和效益，使其他们难以按自由软件的要求和协议全公开、全自由的发布自己的商业竞争优势，也难以让他们全心、全部投入到自由软件事业中来，因此，限制了自由软件快速发展。当然，自由软件的“无私”奉献是人们所期望的。由于这些因素导致了自由软件的快速发展，这时开源软件在自由软件基础上产生了，并逐渐被人们接受。

1.1.2　开源软件定义

开源软件（Open source software，OSS），也称开放源代码软件，是由开放源代码促进会（Open Source Initiative，OSI）定义且提出的，它是一个旨在推动开源软件发展的非营利组织，由 Bruce Perens 和 Eric Steven Raymond 等人在 1998 年 2 月创立，其目的是打算用更符合市场口味的方式来介绍自由软件，试图在商业中找到合适的位置，减少意识形态上的差异，这也导致了与自由软件存在不同的表现方式，即自由软件比与开源软件的定义也严格得多，这导致了诸多 OSI 开源软件许可证不符合自由软件的定义，这与 Richard Matthew Stallman 倡导

① http://opensource.solidot.org/opensource/09/09/25/0554229.shtml

② http://www.fsf.org/licenses/gpl-faq.html

的 copyleft 概念是有区别的，因此，开源软件与自由软件的开源运动中两个不同的表现方式，也使其与自由软件基金会和 Richard Matthew Stallman 分道扬镳。这时，业界就产生了一种自由软件和开源软件两者结合的软件和开源软件（Free and Open Source Software，FOSS）。

开源软件一词出自自由软件的营销活动中。它是一种源代码可以任意获取的计算机软件，这种软件的版权持有人在软件协议的规定之下保留一部分权利并允许用户学习、修改、增进提高这款软件的质量。并且开放源码通过支持源代码的独立同业互查（Independent Peer Review）和快速发展演变提高了软件的可靠性和质量。要通过 OSI 认证，软件必须在获得许可证的情况下发布，该许可证可保证免费读取、重新发布、修改和使用该软件的权利。开源软件是开放源代码开发的最常见的例子，通常开源软件常被公开和合作地开发。

同时，开源软件也是一种软件散布模式。一般的软件仅可取得已经过编译的二进制可执行文件，通常只有软件的作者或著作权所有者等拥有程序的源代码。有时，有些软件的作者只将源代码公开，却不符合“开放源代码”的定义及条件，因为作者可能设置公开源代码的条件限制，因此公开源代码的软件并不一定可称之为开放源代码软件。

开放源码还经常应用于个人、组织和公司的民众运动，寻求将这类软件融入主流应用的方法。根据开放源码促进会及 Bruce Perens 的定义，开放源码由 10 点组成。开放源码并不只意味着访问源代码。开放源码软件的发布（Distribution）条款必须遵从以下标准：

（1）免费重新发布（Free Distribution）：当软件是几个不同来源的程序集成后的软件发行版本中的其中一个组件时，许可证不能限制任何团体销售或分发该软件，并且不能向这样的销售或分发收取许可费和其他费用。即允许获得源代码的人可自由再将此源代码散布。

（2）源代码（Source Code）：程序必须包含源代码，并且必须允许以源代码或已编译的形式发布。如程序在发布时未带源代码，则必须以一种非常公开的方式，在不超过合理重造成本的情况下，让人们获得源代码，例如可以在不收取费用的情况下，放在网络上供人们下载。源代码无疑是编程人员最容易修改程序的形式。不允许故意混乱源代码。也不允许使用中间形式，比如预处理器或转换器的输出。即程序的可可执行文件在散布时，必需以随附完整源代码或是可让人方便的事后取得源代码。

（3）衍生产品（Derived Works）：许可证必须允许修改原产品和衍生产品，并且必须允许在与原始软件相同的授权情况下发布修改过的产品。即让人可依此源代码修改后，在依照同一授权条款的情形下再散布。

（4）作者的源代码的完整性（Integrity of The Author's Source Code）：许可证可以禁止他人以修改过的形式发布源代码，只在该许可证基于修改程序的目的时，才允许随源代码发布“补丁文件”。该许可证必须明确允许发布根据修改过的源代码构建的软件。许可证可能要求衍生产品必须附加不同于原始软件的名称或版本号。即修改后的版本，需以不同的版本号码以与原始的代码做分别，保障原始的代码完整性。

（5）不得歧视任何人或团体（No Discrimination Against Persons or Groups）：许可证不得歧视任何人或任何团体。即开放源代码软件不得因性别、团体、国家、族群等设置限制，但若是因为法律规定的情形则为例外。

（6）不得歧视程序在任何领域内的使用（No Discrimination Against Fields of Endeavor）：许可证不得禁止任何人在特定领域内使用某一程序。例如，不得禁止程序在商业上的应用，或者在基因研究上的使用。即不得限制商业使用。

（7）许可证的发布（Distribution of License）：附加在程序上的权利必须应用于那些使用重新发布的程序的人，无需通过其他人额外加以授权使用。即若软件再散布，必须以同一条款散布进行。

（8）许可证不得专属于特定产品（License Must Not Be Specific to a Product）：附属于程序的权利不得仅限于作为特定软件发行版一部分的程序。如果程序衍生自该发行版并以获得该程序的授权的名义被使用或发布，则使用重新发布的该程序的其他所有人应该享有原始软件发行版本中所授予的那些权利。即若多个程序组合成一套软件，则当某一开放源代码的程序单独散布时，也必须符合开放源代码的条件。

（9）许可证不得对其他软件加以限制（License Must Not Restrict Other Software）：许可证不得对其他随已许可的软件一起发布的软件附加任何限制。例如，不得规定在相同媒体上发布的其他所有程序接受该许可证的限制。即当某一开放源代码软件与其他非开放源代码软件一起散布时（例如放在同一光盘），不得限制其他软件的授权条件也要遵照开放源代码的授权。

（10）许可证必须是技术中立的（License Must Be Technology-Neutral）：任何许可证规定都不可以基于任何单独某项技术或界面风格。即授权条款不得限制为电子格式才有效，若是纸本的授权条款也应视为有效。

而开源软件的许可证有数十种[1]，常见的有 GNU GPL、BSD、X Consortiun、Mozilla Public License、Apache Licence 2.0、Public Domain 和 Artistic 等许可证，这些都是大家认为符合开源软件定义的许可证。

从开源软件定义可明白，开源软件继承、发扬和扩展了自由软件的开放、分享的精神，修正并扩充了自由软件过于严格的定义，从而保证商业软件公司参与和使用开源软件，并能保证自己的效益。同时，从这 10 条定义了也可以明白：

（1）开源软件坚持开放、公开，鼓励通过开源社区参与最大化协作，从而最大限度推动开源软件发展，让更多的人和商业参与这些项目，从而使开源软件的奉献与共享、应用共存。

（2）尊重作者的权利，但不限制源代码自由和共享，同时也保证程序的完整性、优化升级和发展。

（3）保持独立和中立，避免任何可能影响这种独立性的事物。也满足了不同作者的要求，以及商业软件公司自身的利益。

（4）开源软件比自由软件更符合持续发展的要求。虽然无私奉献能让软件更自由、更共享，但个人或小团队的力量难以推动软件持续健康发展，这时就需要商业性的软件公司来推动其发展，但他们最重要的目的是追求必要的经济效益。而自由软件的定义要求是不允许的，因此，开源软件能满足这种要求。

开源软件与自由软件的区别主要表现在：

1．概念不同

符合开源软件定义的软件就能被称为开源软件，而自由软件是一个比开源软件更严格的概念，因此所有自由软件都是开放源代码的，但不是所有的开源软件都能被称为自由软件。但在现实上，绝大多数开源软件也都符合自由软件的定义。比如，遵守 GPL 和 BSD 许可的软件都是开放的并且是自由的。

---

① http://www.opensource.org/licenses/alphabetical

开放源代码的规定较宽松，而自由软件的规定较严苛。很多的开放源代码所认可的授权根本不算是自由软件，所以自由软件不得不和开放源代码划清界线。

2. 表达内涵不同

开放源代码作用是尽可能的使软件最优化；而自由软件则将自由作为道德标准。开放源代码很容易让人简单以为只要把源代码“公开”出来就算是开放源代码了，但是如果用户无法自由运用这些源代码，那么即使公开源代码也没有意义。有的软件公司只是为了想找用户帮它 debug、吸收社区贡献的功能，这样子会破坏了自由软件的原意，也是不道德的，将别人无私奉献的劳动成果收为己呢，这些软件应该遭到谴责。同时，自由软件的愿望就是要给予用户运用软件的自由，这个“自由”就是自由软件的精神所在。但是为了商业化，开放源代码却故意忽略了这个最重要的精神，反而无法让用户体会到“自由”的真意，那么开放源代码这一个替代自由软件的辞句反而把自由的愿望除去了。即在用户使用开源软件时只知道索取，不知道奉献。若是这样只能让开源软件的路越来越窄，或者只能由个人和小团队力量来推动开源软件的发展，但这种发展力量是非常有限，且难以从本质上推动开源软件持续健康发展。

1.1.3 中国开源软件推进联盟定义

中国开源软件推进联盟（China OSS Promotion Union，COPU）于 2004 年 7 月 22 日于北京成立，旨在推动中国的开源软件事业快速、健康和可持续发展。且是在政府主管部门指导下，由致力于开源软件文化、技术、产业、教学、应用、支撑的企业、社区、客户、大专院校、科研院所、行业协会、支撑机构等组织自愿组成的、民主议事的民间行业联合体，非独立社团法人组织。联盟的宗旨是为推动中国开源软件（Linux/OSS）的发展和应用而努力；为促进中日韩以及中国与全球关于开源运动（Linux/OSS）的沟通、交流与合作而努力；为促进全球开源运动（Linux/OSS）做出贡献而努力。同时联盟的作用是为推动 Linux/OSS 的发展，充分发挥联盟在政府与企业之间有关立法、政策、规划和环境建设方面的桥梁、纽带与促进作用；充分发挥联盟在企业与用户、企业与企业、企业与社区、中外企业/社区间、企业与科研、教育、支撑机构之间关于研发、生产、教育、培训、测试、认证、标准化、应用等方面沟通、交流、合作、推进的桥梁、纽带与促进作用[①]。

而 COPU 的 OpenNIC 致力于提供一种免费的网络互通平台，并希望各个服务提供商能够在其上提供相应的接入服务。OpenNIC 并不要求服务商提供免费的服务，而是由于 OpenNIC 使用分布式的架构来设计，因此，希望将来提供收费服务的单位仅象征性的收取运营费用，并标记为基于 OpenNIC。

COPU 为推动开源软件在中国的发展，也建立了适应开源软件发展的开源中国社区，目前由开源中国社区（http://oss.org.cn）、国家开源软件资源库（http://yp.oss.org.cn）、开源技术导航（http://nav.oss.org.cn）和项目协同开发平台（http://matrix.oss.org.cn）等四部分组成。且至 2010 止已经连续五届成功举办了“开源中国开源世界大会”，每一次都来自国内外的著名开源软件专家参会，共商开源软件发展的策略、新方法和新思想。

① http://www.oss.org.cn/?q=node/3

1.2　开源软件状况

开源软件的健康、持续发展，离不开大家奉献，也离不开开源社区相互贡献和程序自愿者勇敢参与。在这种背景下，开源软件经过多年发展，现已是软件产业中一种重要的力量，也得到了 IBM、Sun、Novell 等厂商的支持。开源软件也即将改变软件开发模式，使得聚集大家的力量打破组织边界、持续创造出更高质量、更安全、更易用的软件成为可能。开源软件也会改变软件产业的格局，并会改变软件交互模式，从而导致软件成本迅速下降，世界知识分析机构 Gartner 在 2007 年底就预测，对开源软件兴趣不断提升的有中国、印度和巴西，指出中国与印度市场的扩展潜力最为突出。在 2008 年以后，各大国际软件巨头推出了开源软件战略，而且把中国、印度作为重要战场，不断投入资金、人力、物力，如 IBM 在开放源码方面作出了巨大贡献，奉献了 120 多个项目，耗资数十亿。

早在几年前，Gartner 资深分析员就指出，开源软件可将全球整个软件带出困境，并在很多领域成为应用主流，并使全世界的软件收入逐年增长，从而使开源软件产业值迅速提升。著名 IDC（Internet Data Center）在 2006 年 7 月发布调查报告“全球软件业中的开源软件：市场冲击、破裂和经营模式”：开源软件最终在每一个重要软件领域的生命周期中扮演一个角色，而且将从根本上改变消费者对盒装软件包的价值趋向。IDC 通过对 116 个国家，5000 多个开发者（包括个人和团队）和 38 个开发者网络，以及对比了 OpenCA（主要包括 CA Server、RA Server 和 RA Operator）、IBM、MS Office、MySQL、Oracle、RedHat、SAP（Systems Application，Products）与 Sun 的开源经营模式，并进一步调查发现，至 2006 年 7 月止，世界上 71%的开发者使用开源软件，而且在生产过程中，54%的开发者所在组合或团队使用开源软件，并且全球一半以上的开发者认为今后所在的组织中在开源软件的使用将会不断增长。同时，在 2009 年 IDC 最新一份研究报告显示，在世界范围内开源软件收入每年增长 22.4%，预计在 2013 年达到 81 亿美元，且明显高于 2008 的预测，主要原因包括如下：

（1）调查范围扩大，包括更多的开源软件、更多组织等。

（2）更多的人和组织表现了更高的接受程度，即越来越多的人和组织使用和奉献开源软件。

（3）经济状况加速了开源软件使用的机率。

（4）获得了如 IBM、Sun、Dell、HP、Oracle 等大型软件巨头积极支持，这也成为开源软件被接受和采用的主要原因之一。

（5）很多开源软件逐渐形成了商业模式，更多软件厂商提供了更多的开源软件。

（6）开源软件在各方面应用提高了竞争优势。

同样，开源软件在中国也逐年被使用，正如倪光南院士在 2006 年所说，开源软件在中国有多方面的发展机遇：第一方面是国家战略方针的建设，第二方面是信息安全要求越来越高，第三方面是软件正版化，第四方面是各应用软件的改变，第五方面包括国产开源软件的成功应用。其实我国早在 1999 年 7 月 15 日，原信息产业部曲维枝副部长主持召开了“Linux 与中国软件产业研讨会”，这是中国政府首次明确支持以 Linux 为代表的开源软件。此后，开源软件在中国发展迅速，并且在中国的科技计划大力支持基于开源软件的各种基础软件，这些基础软件包括操作系统、数据库管理系统、中间件、办公软件等。在《国家中长期科学和技

术发展规划纲要（2006—2020年）》中，又将基础软件纳入信息领域的“核心电子器件、高端通用芯片及基础软件”重大专项之中，其中有相当大的部分是基于开源软件的，这为今后15年里开源软件在中国的发展铺平了道路。最近，科技部继续将基础软件作为“十二五”规划中的重要课题（出自《科技日报》2010年7月14日第011版）。这些政府策略将进一步有效改变2006年倪光南院士所提出的中国开源软件面临的挑战：

1. 缺乏有效的市场支持

由于最初大部分开源软件都是国外个人或团队奉献的，因此开源软件在中国市场迅速发展在不同程度上受到了影响，直接导致开源软件在中国市场还没有经历“市场可用性”的考验。但近年来，开源软件已经在国内快速发展，也得到大多企业、个人和团队接受。

2. 起初国内开源社区薄弱

由于开源社区是促进开源软件发展有效方法之一，而中国的开源社区与国外的开源社区有相当一段距离，有效的开源社区也很少，造成这种原因主要缺乏支持开源软件的基金会，而且目前支持科技的基金会很难支持开源软件，只能通过申请项目给予支持，这难以满足开源软件的自由共享真正的含义。但中国近年来提出自主创新，为开源软件创造了良好的环境。

3. 国内开源软件垄断情形较突出

由于目前国内很多领域都是使用的商业型软件，已经形成了一个“定性”思维难以用新的东西，因此，在起初很难推广开源软件，但近几年来，随着开源软件的推广和逐步被接收，开源软件已在很多领域得到了应用。

在我国的开源软件的发展情况表明，这些挑战还将持续下去，并且在一定的领域也将阻碍开源软件在我国的发展。毕竟国内开源的流行程度还未深入到个人和团队，即国内的个人或团队对开源件的索取大于奉献。

1.2.1 开源软件发展状况

随着信息科技时代快速发展，尤其是在互联网的兴起，开源软件在整个软件业起着巨大推动作用，并且开源软件在世界范围内已经广泛地融入商业软件，并被各国政府、企业、机构广泛的采用。同时，开源软件在全球范围还起着打破垄断的巨大作用，使普通用户业能真正享受到物美价廉的通用产品，因此也赢得了世界各国政府以及各界人士越来越多的支持，对打破软件垄断的边界起到推动作用。

1.2.1.1 开源软件的发展状况

自1998年11月开源运动促进会（ISO）正式成立，起初的目的是让业界注意自由/开源软件，并在自由软件运动的“对抗”态度之外开辟另一条道路。但开源软件的发展需要创新奉献和开源社区两个主要条件来促进。

1. 开源软件需要创新和奉献

2007年5月15日，以“开源与创新交流与奉献”为主题的“首届OpenOfice.org国际高峰论坛”在北京召开，该会由COPU主办，北京红旗中文2000公司承办，并在中方的邀请下，全球OlaenOfice.org开源社区负责人Louis Suarez—Ports先生、OpenOffice.org项目总负责人Michael Bemmer先生来京参加了“首届OlaenOfice.org国际高峰论坛”。论坛最后决定对

国家倡导的自主创新，全面发展的精神，秉承“开放、自由、创新”的宗旨，推动开源社区产业的发展。进而从这次峰会容易得知，开源软件可持续发展需要创新和奉献，而不是过多的索取。

2．开源软件需要开源社区

2007 年 6 月 21 日在中国广州召开了“2007 开源中国，开源世界”高峰论坛，Apache 创始人 Brain Behlendor 第二次来中国，并在论坛圆桌会议上表示：“开源仍能改变世界”。这就使得开源软件成为全世界计算机/软件行业的一种发展趋势。Intel 的 Dirk Hohndel 和其他专家均认为，市场的潜力和需求是开源运动发展的主要推动力。同时，Dirk Hohndel 也认为，开源运动的成功来自开源社区，但支持开源社区开发的志愿者 85%来自大公司的程序人员（而不是来自社会），不同国家由于文化、语言差异，使开源运动的合作受到影响。同样，Oracle 的 Wim Cockaerts 说，开源运动的精髓是社区开发机制，社区把开发的程序放到互联网上，争取全球广大志愿者对程序的修改、反馈。源程序代码数量很大，命令行数目很多，缺陷也可能很多，修改工作量很大，只有发动全球志愿者，才能做到及时修改、完善，保持很高水平。

Google 的 Chris Dibone 说，作为用户来说，开源软件给予选择权、控制权，自己可进行修改，调整（对软硬件进行最佳配置），这是私有商业软件做不到的，当然不是说涉及到核心技术的所有程序都要开源的。同样，Sun 公司的 Simon Phipps 说，开源改变了软件的开发方式，是合作创新和自主创新的完美结合。同时，Red Hat 的 Tom Rabon 进一步也指出，开源运动的发展阶段分为：1995—2001 年是推进全球化，2001—2006 年是推进信息化，2006—2011 年主要是创新。开源软件的发展是离不开一个开放、奉献、创新的开源社区。

关于开源软件社区的特征，COPU 秘书长陈首群在 2007 年第 07 期《信息系统工程》提出了开源社区 10 个特征：

（1）开源社区一般由志愿者参与开发。不需成本（如上所述，目前在社区核心层中出现了一批由领取工资的固定人员所组成的骨干核心层。这时当然要增加一些成本，但总的来说社区还是低成本开发）。

（2）由志愿者组成的开发人员来自全球各地，其中不乏高水平者。据统计，社区的志愿者 70%～80%来自企业（而非社会）。

（3）社区开发人员具有自由、开放、共享、选择、透明、协同、奉献、无偿的理念，并不以追求商业利益为其目的。

（4）开源社区开发的开源软件，实现了其全部性能的创新工作（原创性），社区开发的这些性能、技术，可能还不够稳定、高效、优质和成熟；在此基础上推出社区版（或β测试版或参考平台）；发布社区版时，其全部源程序代码将在网上公布。人们可从网上免费下载社区版。

（5）社区开发是采用集体开发、合作创新的方式，在这里或这个开发阶段不存在所谓自主开发创新的空间。

（6）在社区开发阶段，在开发成果（推出的社区版）中，一般也不存在技术秘密和商业秘密。

（7）社区开发人员没有明确的研发路线图（因为社区开发人员并不了解用户的需求，很难自动制定出完整的研发计划）。

（8）社区开发人员不必对版本的挑错纠错（Bug Fix）、打补丁（Patch）、解决问题和质量

认证等工作负责．社区只负责对投递给社区的有关缺陷（Bug）、补丁（Patch），进行有选择的验证工作。

（9）社区开发人员不需对市场策划、产品销售、技术支持以及产品服务负责 不需对后续开发、产品发布、商业模式负责。

（10）社区开发只完成了开源软件产品开发全过程中极端重要且必不可少的先导开发阶段（开发“创新技术”，开发全部产品性能）；作为开源产品的开发全过程，还需要企业（非社区）开发阶段（自主开发成熟技术，优化产品性能，实施质量认证， 提高”产品质量）与之衔接，即要求后续的企业开发阶段相应跟进。

1.2.1.2 自由与开源软件（FLOSS）发展状况

2008年12月02日在法国巴黎召开的“开放世界论坛（OWF）”大会上，会议组织者披露了一份到2020年开源软件产业发展规划路线图，该路线图集全世界顶级的FLOSS专家、学者和投资者（包括来自20个国家和160名成员和1200位热心的国际听众）对FLOSS相关的趋势进行了预测，并分析和比较了这些发展趋势的风险提出建议以减少FLOSS所带来的风险。这份报告名为“2020 FLOSS Roadmap”，如图1-2所示。它描述了2020年前开源软件的瑰丽前景，公布了80项开源产业的发展建议。并指出2020年开源软件将进入主流软件产业。虽然云计算、绿色计算、环境计算、移动机器人等技术将对网络社会产生深层次的影响，同时，随着社交网络的普及，云计算服务不仅能够人与人之间的互动，还将推动企业或政府之间的互动，加之近年兴起的物联网更能加深其应用。但越来越多的CIO将会选择FLOSS软件，这些具有低生态影响特征的软件将成为绿色数据中心和其他商业模式的核心内容。

图1-2 2020年开源软件发展路线图（来源：开放世界论坛）

并且报告还预测，FLOSS社区将在未来演变为一种业务生态系统，将成为实现企业开放IT的战略工具，未来将有40%的IT职位与FLOSS相关。同时报告还强调。云计算将成为一种无所不在的计算机模式，未来企业将基于“开放云（Open Cloud）”实施云计算，以构建其信息系统的主要架构。

当然，报告也阐明为在2020年实现路线图也蕴涵风险，云计算和Web服务的应用也同

样存在一定风险。当软件被隐藏真情起来，只以界面的形式展现时，这种新的方式将限制用户对应用源代码的查看，这也导致 FLOSS 的许可走向两个极端，要么毫不相关，要么进行太多无意义的限制。该份报告将自由与开源软件（FLOSS）进行结合披露，这就充分说明自由与开源可以共存，可以共同推动软件业发展，这是因为：

（1）获得了 FLOSS 的许可，一般就具备自由和开源双重标准。可以任意使用开源代码，且还赋予研发人员研究、变更、改进软件的权利。

（2）在第一类主要软件的生命周期中，FLOSS 已经或者必将推动软件快速发展。

（3）FLOSS 弥补了自由与开源之间的差异和相关许可证不兼容等问题。

同时，FLOSS 发展路线图从 Internet、信息社会、产业和研究与开发四个层面预测了 2008 年—2020 年 FLOSS 发展趋势（图 1-2）。并且在报告中也给出了 FLOSS 发展的预测和建议[2]。

这些预测主要包括：由于 FLOSS 具备了自由与开源的特性，同时还包括了自由、参与、奉献、创新和沟通等特征来创造一个更公平的数字世界，从而有助于缩小全球数字鸿沟和为缩小贫富差距作出贡献。并且 FLOSS 将成软件研发的主流技术，范围包括从基础设施到应用，作为软件产业若干环节的标准而占据支配地位，其开发模型将被 IT 解决方案供应商和用户广泛采用。同时，FLOSS 通过社区将催生一代商业生态绝色系统，出现开放模型、闭源模型和开源软件模型共存的模式进行广泛应用，并且通过云计算服务实现人与人之间，推动企业与政府、人与企业等之间的互动。

这些建议提出的前提是在一个稳定的、清晰知识产权制度和许可证下，尽可能从自由和开源软件的自由、共享、奉献、创新等角度提出 FLOSS 发展建议。具体如下：

（1）鼓励 FLOSS 社区间的交流，也通过开源社区促进 FLOSS 发展，努力综合自由和开源的优势。

（2）鼓励用户、程序员和自愿者作出贡献，包括相关税收优惠。

（3）努力提高 FLOSS 的使用率，同时，提高 FLOSS 使用的质量。

（4）将开放性作为创新和商业生态的标准，并建立基于开放标准和开放服务建立 FLOSS 平台。

（5）大力发展 FLOSS 教育，在大学及其他教育机构中建立 FLOSS 意识。

（6）投资 FLOSS 的研发以发展战略性技术和服务。

1.2.1.3　开源软件使用的区域差异

2010 年 4 月英国著名调查机构 451 Group①的 CAOS（开源软件的商业化应用）机构的服务经常被用来比较自北美、欧洲、亚洲和南美用户在使用开源软件态度差异[3]。

（1）与亚洲、南美洲、欧洲相比，北美地区的受访者在考虑是否采用开源软件的时候更加看重成本节约因素，而在亚洲和南美洲，尤其是在欧洲，节省成本的因素并不是人们是否采用开源软件的重要驱动力。

（2）与其他地区的受访者相比，更多的欧洲受访者把具有更大的灵活性作为采用开源软件的主要驱动力，而亚洲地区的受访者比欧洲、美洲地区的受访者更关心是否有垄断的供应商。

① http://blogs.the451group.com/opensource/2010/04/22/flee-451-group-report-on-regional-differences-in-attitudes-to-open-sourc-adoption/

（3）亚洲地区的受访者把降低成本列为他们最青睐的开源软件带来的好处，而南美、欧洲和北美地区的受访者则把更大的灵活性作为他们最欢迎的受益。

（4）南美洲的受访者似乎是从减少垄断供应商的过程中获得收益最少的。与南美洲、亚洲、欧洲地区相比，北美的受访者实行开源使用政策的倾向最弱。

（5）同样，北美地区的公司最不可能跟踪开源软件的部署情况，也最不可能跟踪开源软件在开发项目中的使用情况。同时，北美和南美地区的公司在制定促进开源项目发展的政策方面比较落后。

（6）超过 50%的亚洲受访者称考虑到今年的经济情况，他们更可能采用开源软件。紧随其后的是北美、南美、欧洲（少于 40%）。

1.2.1.4 中国开源软件发展状况

开源软件已在全球高速发展，但在过去 10 多年中，开源软件在中国的发展并不是很理想，与国际上其他国家相比还存很大的差异。但开源软件在中国的发展，从最初的萌发期，到青春期，再到转折期，到现在的发展期。这也不能影响开源软件现已成为一个软件产业，特别是互联网技术成熟和发展，为开源软件发展提供了重要的平台。著名开源件网站 SourceForge 在 1999 年还只有数百个开源项目，到 2008 年初其开源项目数已经超过 17 万个，到 2010 年开源项目已经达到了近 20 万个，几乎覆盖了软件应用的所有领域。但实际上，中国的开源却是另一种情况：社区冷、企业热、使用热、开发冷，但在 2010 年时，这种现象有所改变，特别是社区有所起温、企业继续热、使用多奉献少，开发逐热。但与国际上开源软件发展相比，还是存在很大的距离。使用开源产品的公司和人员众多，但开源社区的发展并不很理想，真正参与开源产品开发和社区奉献的开发人员不多。《程序员》杂志在 2008 年总结以下几个方面的原因：

（1）语言的障碍，阻碍了中国软件开发人员参与国际开源社区。

（2）东西方文化的差异以及对开源文化的了解不足。

（3）经济上的快速发展带来的工作和生活压力。

（4）中国软件开发发展的时间还不长，核心开发人员的积累还不够，缺乏开源关键人物。

（5）大学教育在开源领域严重不足，教师也缺乏了解。

同时，中国开源软件还需要解决以下问题：

（1）缺乏多元化发展。国外开源软件可谓是百花齐放，比如 Apache、MySQL、SugarCRM 、JBOSS、Compiere、Eclipse 等等都有不同程度的发展，而国内市场有八个 Linux 厂商在发行各自的 Linux 版本，同质化严重，与用户的实际需求尚有距离。因此，有必要尽快把注意力从操作系统方面转向应用程序和系统程序。

（2）如何保持开源社区可持续发展。中国关于开源方面社区也是比较多的，但是规模都不大，难题在于如何整合资源，让社区进一步发展。

（3）社区如何和国际社区紧密保持联系，实现技术同步发展。在知名的国际社区中，来自中国企业的贡献比较少，而且中国所有的开源厂商目前尚不能影响国际开源领域的发展潮流，多数只是在跟随，更不能说引领了。

（4）如何实现开源软件发展的商业模式。发展开源软件，现在普遍接受的是免费购买软件、服务付费，那么，还能发展其他的模式吗?这是开源界应该考虑的问题。

（5）开源软件与资本市场脱节。可以说是开源软件与市场和用户的脱节，导致了资本的脱节。另外，与传统软件相比，开源软件更需要融资中介机构，但是国内还没有人来做这种工作。

但开源软件给中国的软件产业提供了加入国际软件极好的机会，为中国软件业快速发展提供帮助，且使研发人员能够参与开源社区和产品开发，从而有效打造中国开源社区的良好生态系统具有重要的意义：

（1）开源软件集合了全球软件技术发展的精华，可以让研发人员和软件企业充分学习和吸收。

（2）开源软件覆盖了软件应用各个领域，可以中国的软件企业在此基础上发展应用。只要遵守其商业规则，就能创造出商业价值。

（3）作软件的用户会都会更愿意自己用的系统是开源的，这样便于维护升级、系统整合和安全。

（4）可以逐步使中国研发从最初对开源软件的索取到现在创新和奉献，为开源软件在中国快速发展和在世界中协同进步创造条件。

总之，随着开源软件在全球范围内发展，必将推动软件产业进步，也将带动中国软件业的发展，但要使开源软件在中国持续、快速发展，还得具备这些条件：需要营建好的环境和获得国家的相关政策；加强开源社区的建设和引导，用多种策略培养人才，努力推广开源软件项目，继续推广中国开源和应用模式。

1.2.2　开源软件应用状况

开源软件的共享和自由，以及发展最终的目标就是实现有效应用，在推进开源软件进步的同时，能创造商业价值，并在一定范围内，推动软件产业进步，实现有效的经济效益，提高研发人员开发效益。目前已经在电子政务、教育教学、中小企业信息系统集成、图书馆数字化等领域得到了有效的应用[5]。

1.2.2.1　开源软件在经济建设上的角色

开源软件改变了未来软件的开发模式，使软件研发更容易，能使更多的人和团队使用和奉献开源软件，也会逐渐改变软件产业的商业模式和经营模式。使得聚集大家的力量打破了组织边界、持续创造出更高质量、更安全、更易用的软件成为可能。更重要的是改变了软件的使用方式——从“使用许可”为主商业模式变成以支持、咨询等面向服务为主的商业模式，使得在全球服务经济转型的过程中扮演着日益重要的角色。根据 2006 年欧盟首个开源研究报告《开源对欧盟软件通信产业竞争力和创新的影响》指出：截止 2006 年底，全世界接触和应用开源软件的企业占到了总数的 50%以上，特别是美国则高达 80%～90%，在 2004—2006 年间越来越多欧洲企业都开始使用开源软件，2005 年底已经超过 40%，主要分布在电信行业、媒体行业和公共管理部门。截至 2006 年底，欧盟企业大约投入了 12 亿欧元进行开源软件的开发，为市场提供了 56.5 万个就业机会和 2630 亿欧元的相关收入。在 2007—2010 年间，将有 95%的全球 2000 强企业广泛采用开源产品和服务，开源软件将在 2007—2012 年间内占据传统软件市场 22%的份额。到 2010 年，欧盟开源相关的服务将占到其 IT 服务产业产值的 34%，

开源软件带来的产值将占欧盟 GDP 的 4%，超过欧盟整个软件行业产值的 25%。而 2010 年来自 Zenoss 的调查报告显示：

（1）有 98%的企业使用开源软件。

（2）大开公司有 78%的人趋向使用开源软件。

（3）自 2006—2010 年间，开源软件使用的增长比例分别为：2006 年 26%，2007 年 38%，2008 年 48%，2009 年 71%。

同时，也指出了对选择开源软件的原因：

（1）认为开源软件文档差，难获得及时且有效的技术支持。

（2）服务支持保证是企业选择专有软件的第一原因，即开源软件没有一合适、满足用户品味的服务支持体系。

在 2008 年，著名市场调查机构 Gartner 对亚太、欧洲以及北美的 274 名终端用户进行了调查表明：有 85%的公司已经开始使用了开源软件。到今年至少有 80%的商业软件解决方案将包含实质性的开源软件。49.7%的开源软件应用是用于重要任务的应用程序。相比之下，专有软件有 59%用于重要任务应用程序，有 58.8% 用于内部开发。

在操作系统方面，根据 IDC 的报告，Linux 的市场整体营收与 PC 服务器和套装软件硬件预计在 2008 年将达到 35.7 亿元，年均增长 26%。Linux 服务器在全世界占有率预期从 2004 年的 20%增长到 2008 年的 27%，在美洲的份额将从 24%增长到 32%，在欧洲将从 2004 年的 16%增长到 2008 年 25%。而在 Linux 桌面系统的使用占用率方面，欧洲高于美洲：2004 年，欧洲 5%，美洲 2%，预计 2008 年，欧洲将达 9%，美洲为 4%。

在中间件方面，2006 年 Apache 在欧洲市场占有率为 84%，在美国的市场占有率为 66%，根据 BZResearch 的调查，2004 年，JBoss 是应用服务器中间件市场的领导者。2005 年，Jboss 应用服务器的市场份额为 37%，而作为 2005 年应用服务器市场的领导者，IBM 则仅拥有 37.2%份额，而目前市场份额最重要的就是 Tomcat 服务器。

在数据库方面，根据 IDC 的数据，2002 年，西欧 33%的公司使用开源数据库软件，而个人方面只有 11%的用户使用开源数据库软件，而到了 2005 年，这两个数据分别为 25%和 33%。另外，据 Gartner 的统计分析，比起整个数据库市场 8%份额，开源数据库软件的使用率在 2005 年增长了 47%以上。

在桌面办公方面，据 Openoffice.org 的报告显示，到 2006 年中为止，Openoffice2.0 在主要的下载点被下载 62500000 次。据 Forrester Research 统计，2004 年，Openoffice 在北美大公司的市场占有率为 8.5%，在欧洲，仅德国企业中的市场占有率就达到了 8%。

1.2.2.2 开源软件商业模式

开源软件的发展主要集中在开源社区、开源软件质量和开源软件应用推广，以及怎么在应用推广实现有效的商业模式。一件开源软件的质量好坏、效果如何，只有通过使用后才能给出一个合适的评价，而这个评价往往又来自开源视社区。因此，探索开源软件的商业模式对促进开源软件的推广应用具有重要的意义。Eric Raymond 所提出开源的软件交付模式，到今天已形成了一条走向成熟的软件产业生态链，在这样的生态链上如何找到自己的价值，如何开发商业模式，成为今天使用开源软件最关心的问题之一。

1．开源软件的商业化历程

Eric Raymond 从概念和理论上理清了自由软件到开源软件的“自由与商业”的障碍，使开源软件能在商业应用中不受到严格许可证和知识产权的约束，并且又能做到源代码的开放和共享。而开源软件的商业化历程大致可以分为以下几个阶段：

（1）萌芽阶段（1995 年以前）。在 1995 年前，相关研发人员已在开源社区开始出售一些与开源软件相关的物品，但当时并没有引起开源社区的人注意。但直到 1995 年红帽软件公司正式成立之，才逐渐引起大家注意。

（2）探索阶段（1995—2001 年）。1995 年，Young 购买了 Ewing 的股份，把新公司命名为红帽软件，同时发布 Redhat Linux2.0。红帽公司的成立，拉开了开源软件探索商业运作的序幕。在这种情况下，越来越多的厂商开始在开源件上寻找商机，进一完善开源软件的文档和提供相关的技术支持。仅在 1999 年至 2000 年，就有红旗 Linux、中软 Linux、蓝点 Linux、冲浪 Linux、TurboLinux、TomLinux 等品牌软件公司出现，但这个阶段并没有找到合适的开源软件商业模式，大多都停留在探索阶段。

（3）发展阶段（2001—2003 年）。这个阶段是开源软件商业模式的优化阶段，试图尝试使开源软件走向商业模式的各软件公司因为网络泡沫的破灭，让很多开源软件企业要么倒闭，要么退出市场，只仅存如 Jboss 和 MySQL 少数公司在商业模式的优胜劣汰中获胜。

（4）融合阶段（2004—2006 年）。这个阶段是开源软件走向应用的主要时期，这是因为开源软件实现的功能相对单一，这时就需要将多种开源软件进行融合集成，从而真正体现开源软件的开放和自由。在这种情况的驱使，以及竞争角逐和要求下，多家有名的软件企业相互收购，出现如 2003 年 11 月 Novell 收购了排名全球第二的 Linux 发行商 Suse，IBM 于 2005 年 5 月收购开源软件 Gluecode，以及 2009 年 4 月 Oracle 收购 Sun 等。

（5）应用阶段（2007 年—）。这个时期是开源软件商业模式正式确定后的第一个正式应用阶段，各种有影响的开源软件相继推出并应用到各个领域，同时，为实现更大功能，相关开源软件也进行有效的整合应用，如面向 Web 的 LAMP[Linux、Apache、MySQL 和 PHP（或 Perl、Python）]、SSH（Struts、Spring、Hibernate）等。

2．开源软件的商业模式

经过十几年的发展，开源软件的商业模式已逐渐成形，常见有以下几种模式：

（1）软件免费，服务收费。由于受到一些严格的开源软件许可协议限制，开源软件在授权转让时不允许收取费用，因此在开源软件协议和定义的允许范围内，出现一种订阅（Subscription）服务模式来进行服务收费，如 Redhat 公司收购 JBoss 之后，使 Redhat 的产品得以扩展，由此使 Redhat 公司可以得到更多的订阅服务，从而收取更多的服务费用。这种模式的典型代表还有 SourceLabs 公司和 SpikeSource 公司，这两家公司并不主推自己的产品品牌，而是与多方开源软件厂商或社区合作，利用他人提供的开源软件，从而提供技术测试、集成、维护等服务。

（2）双授权模式。当软件厂商拥有该软件的所有代码所有权（开源代码也是有版权的），那么该厂商就有可能采用双授权的模式（既针对不同用途的用户采取不同授权协议）去对商业用户收取授权费用。与此同时，产品仍能够融入开源社区中，以获得改进信息、得到开发者的支持、赢得口碑、增加用户基数，进而占领市场，增加收入。如 MySQL、Qt 等知名公司就是采用双授权模式实现商业效益的。2005 年 Sun 曾斥资 41 亿美元收购了 StorageTek 公

司，2008 年 1 月 MySQL 被 SUN 以 10 亿美元收购，Oracle 出资 85 亿美元收购了中间件厂商 BEA，在 2009 年 4 月 Oracle 以 74 亿美元又收购 SUN，更是创下开源企业被收购的记录，从而为公司占据更多的市场，并获得更多的利润。

（3）免费知识，咨询、培训收费。开源社区中有相当部分知识并非针对普通用户，而是针对开发者的各种开发工具、库以及框架。并以此提供一些免费知识供用户参考，但要获得有进展的知识和技术咨询时，就需要收取一定的咨询和培训费，以及通过将这些知识和文档汇编成书籍进行出售。如在开源社区 Java 中大热的 Spring，凭借自身强大的技术实力，以压倒性的优势占领了 J2EE 的 framework 市场，然后依靠咨询与培训收取费用。开源软件出版商 O'Reilly 公司组织各种开源软件会议，推进开源理念和开源软件技术的传播与发展，以出售书籍赢利。在中国也有这种模式存在，如中国的即时科研集团目前也在大力开展 Linux 培训。

（4）通过 Internet 提供免费软件，以广告收费。广告模式是通过 Internet 网提供免费的软件，并以此授受相关广告进行收费，因此目前有不少基于互联网应用的开源软件采用此类盈利模式。如 Mozilla 通过与 Google 的合作，依赖惊人的下载量与使用量，Mozilla 基金会每年能获得数亿美金的收入，远远超过国内大部分软件公司或是互联网公司的盈利。与此同时，Mozilla 所构建的插件体系，不仅养活了自己，同时也养活了一批中小企业与个人，由此形成一个完整的浏览器产业。当然这种模式对促进开源软件的进步和推广也起到巨大的作用，但过多的、繁杂的商业广告出现在开源软件中也是让人心烦的，因此，要使得软件免费与广告长期并存，还需要进一步讨论具体的实现方案。

（5）免费软件，收费硬件。IBM、SUN、惠普等公司，在开源软件领域投入巨大，但这一切并非是做善事，它们可以从配置了开源软件的硬件中获取巨额回报。但也促进了开源软件进步，毕竟在当前经济时代，没有一定的费用怎能推动开源软件事业的发展，虽然有相关的基金会提供一些帮助，但开源软件实质性的发展和进步需要得到大型软件公司的研发人员加入才能有效的得以实现。在中国就是因为在一定范围内缺少了有影响的基金会和大型软件公司的支持，所以开源软件在中国的发展以索取为主，奉献为辅，当然近年来这种现象有所改观。

（6）免费社区版，收费企业版。对于一些通用软件，如操作系统、数据库软件和软件体系架构系统，开源软件厂商一般采用针对不同用户，提供不同版本的方式，如 J2EE 提供了标准版和企业版；又如 MySQL 产品就同时推出面向个人和企业的两种版本，即开源版本和专业版本，分别采用不同的授权方式。在这种模式中，利用免费版本软件为赢利的收费版本创造或维持一种市场地位，这种模式较为普遍。

（7）开源软件，商业软件。将免费的开源软件与可赢利的商业软件捆绑销售，以开源软件带动商业软件的销售，这也是一种不错的商业模式。如 Novell 就将自己原来丰富的中间软件和应用软件迁移到 Linux 平台上，通过与 Linux 捆绑，为客户提供高价值的综合解决方案。红旗和 Turbolinux 也在积极加强与应用软件厂商的联系或自己开发商业软件，通过附加更多的商业软件来增加收入。

（8）Ubuntu 模式。Ubuntu 是一个以桌面应用为主的 GNU / Linux 操作系统，它的目标在于为一般用户提供一个最新的、同时又相当稳定的主要由自由软件构建而成的操作系统。Ubuntu 具有庞大的社区力量，用户可以方便地从社区获得帮助，Ubuntu 是由 Mark Shuttleworth

创建的，且他始终认为 Ubuntu 是自由的。普通的桌面应用版可以获得 18 个月的支持，标为长期支持版本（Long Term Support，LTS）的桌面应用版可以获得更长时间的支持。Ubuntu 的所有发行版本都可以免费获取。除了可下载光盘镜像文件（CD Image）外，用户也可通过邮寄服务免费获取安装光盘，但是，一些用户难免会在使用过程中遇到自己无法解决的技术问题，这时 Ubuntu 就可以为这些用户提供收费服务的技术支持。目前 Ubuntu 是在 Canonical 公司下提供技术支持并收费服务费用，但收费标准是全球统一的，收费标准为如下，但都包括 17.5%的附加税。

若提供每周五天从早上 9：00 至 17：00 的技术服务，桌面版一年收费标准为 144.74 欧元；服务器版一年收费标准为 434.21 欧元。

若是每周全天服务，桌面版年收费标准为 521.05 欧元；服务器版收费标准为 1870.72 欧元。

Ubuntu 版本的变化从 2004 年 10 月 20 日至 2010 年 10 月 10 日至共发布了 13 个版本，其版本变化时间线如图 1-3 所示。

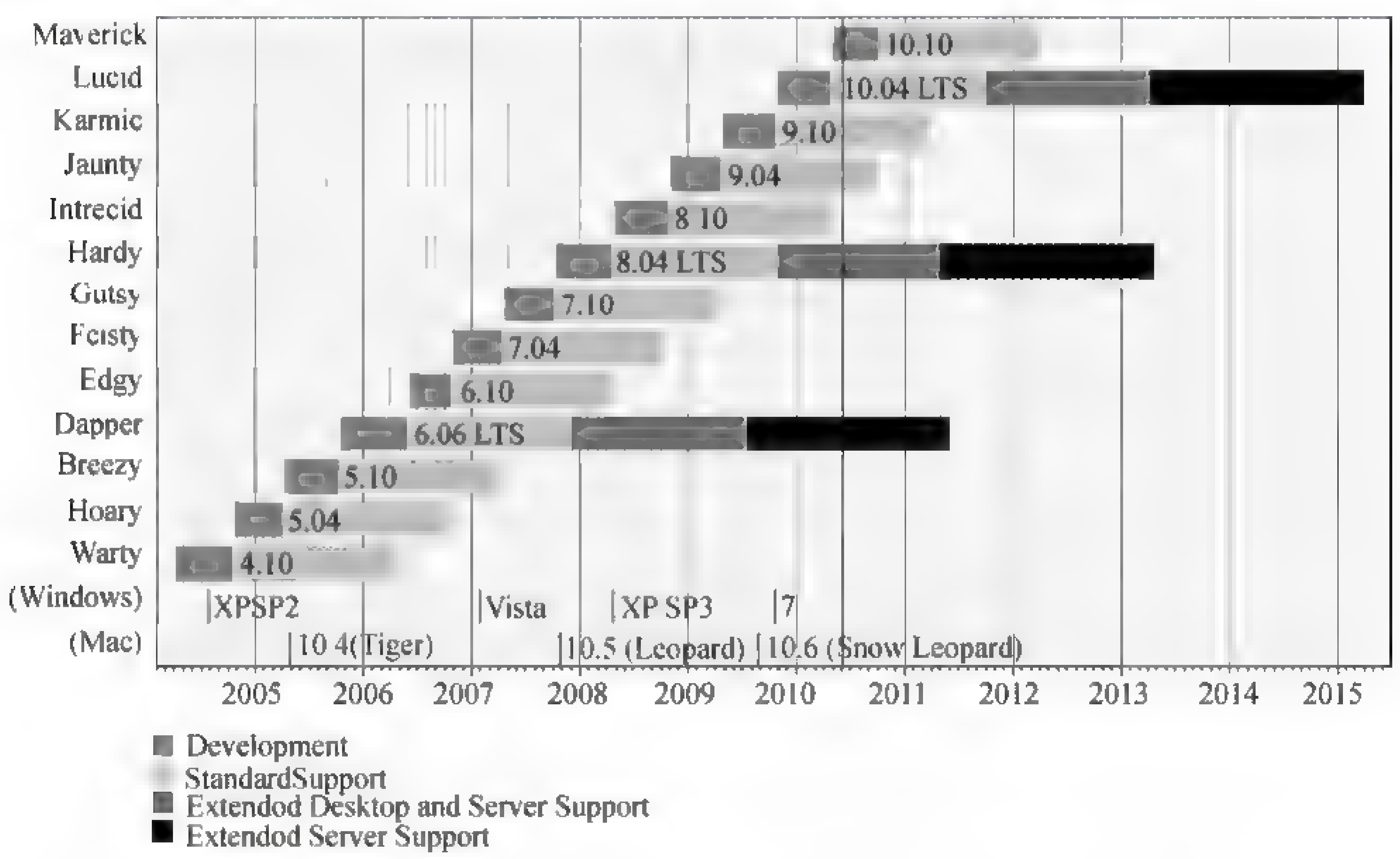

图 1-3　Ubuntu 的版本变化图（该图更新时间为 2010-06-11）
（来源：http://zh.wikipedia.org/wiki/Ubuntu）

总之，开源软件的应用已经深入到了各个领域，且这些领域带来了极大的利益，也推动了这些领域发展。且随着不同的商业历程发展，以及不同的商业模式的应用，使开源软件从单纯的开放、自由和共享逐渐走向商业化道路。

1.3　开源软件分类

软件分类从软件表现方式上分可以分为自由软件、闭源软件（或专有软件）、商业软件、

免费软件、开源软件和共享软件（或试用软件）；从软件结构上看分可分为系统软件、应用软件、专用软件。而开源软件的分类按中国开源社区[①]对开源分类共可分为编程语言、程序开发、开发工具、jQuery 插件、建站系统、企业应用、服务器软件、插件和扩展、数据库及相关、应用工具、游戏与娱乐、管理与监控、操作系统及相关等十三大类。开源软件典型分类名及代表如表 1-1 所示，常用的开源软件工具见表 1-2。

表 1-1　典型开源软件分类名及代表列表

序号	开源软件分类名	典型代表
1	应用服务器	JBoss、Apache Tomcat、Geronimo 等
2	消息服务器/消息平台	Zend（PHP）、Zope（Python）、Exolab Group（J2EE，集成 OpenEJB，OpenJMS，OpenORA，Tyrex）等
3	企业资源计划	Sajix（Java）、Compiere 等
4	客户关系管理	SugarCRM、Compiere、OpenERP 等
5	项目管理	Trac 等
6	内容管理	Drupal、Joomla!（PHP）、Plone（Python）、PostNuke（PHP）、XOOPS（PHP）等
7	人力资源管理等	PhpGroupWare 等
8	云计算	abiCloud、Eucalyptus、Hadoop 等
9	库存管理	Akopia 等
10	B2B、B2C	OpenApplications、ebXML 等
11	图书馆资源系统	Evergreen 等
12	Blog	WordPress（PHP+MySQL）、lifetype、b2evolution 等
13	无线	Wapuniverse 等
14	论坛（BBS）	phpBB、BMForum、PHPWind、动网论坛等
15	数据库	MySQL、Apache Derby、PostgreSQL 等
16	桌面类	GNOME、KDE、Xfce、Ximian、LXDE 等
17	窗口管理类	Blackbox、Enlightenment、Fluxbox、IceWM 等
18	CD 刻录类	Brasero、K3b、GnomeBaker 等
19	办公软件	OpenOffice.org 等
20	PDF 软件	PDFCreator、Evince、Sumatra PDF 等
21	金融软件	GnuCash 等
22	电子邮件	KMail、Mutt 等
23	文件传输软件	FileZilla、Samba、WinSCP 等
24	P2P 共享软件	Transmission、BitTorrent、eMule 等
25	远程登录	RealVNC、TightVNC、UltraVNC 等
26	数学软件	Scilab、LaTeX、GNU Octave、Yacas 等
27	图像类	OpenCV、Blender、Crystal Space 等
28	安全类	Open Antivirus、GnuPG、OpenSSH、Putty、Password Safe 等
29	操作系统类	Unix、FreeDOS 、Linux、ReactOS、GNU Hurd 等
30	商业智能相关软件	Pentaho、Kettle、Mondrian、Jpivot、BIRT、WEKA 等

① http://www.oschina.net/project/tags

表 1-2　常用的开源软件工具一览表

序号	软件名称	简单描述	网址
1	Linux	操作系统内核程序	http://www.linux.com/
2	FreeBSD	著名的 BSD 操作系统。特点是 TCP/IP 协议的实现十分的稳定，常作为互联网上的服务器使用	http://www.freebsd.org/
3	Perl	一种有很强的数据处理能力的编程语言	http://www.perl.org/
4	Gtk	一种跨平台 C 语言图形用户界面工具包	http://www.gtk.org/
5	OpenOffice	著名的开源 Office 办公软件	http://www.openoffice.org/
6	Gimp	一个图形图像编辑软件	http://www.gimp.org/
7	Jabber	一种开放源代码的以 xml 为基础的及时通信协议	http://www.jabber.org/
8	Apache	开源界最著名 Web 服务器的产品之一	http://httpd.apache.org/
9	Qt	一种跨平台的 C++编程平台	http://qt.nokia.com/products/qt/index.html
10	Firefox	著名的 Web 浏览器	http://www.mozilla.org/products/firefox/
11	Pentaho	一种著名的商业智能平台	http://www.pentaho.com/
12	Kettle	一种 ETL 工具	http://kettle.pentaho.com/
13	Mondrian	一种 OLAP 工具	http://mondrian.pentaho.com/
14	Jpivot	JPivot 是一个 JSP 自定制的标签库，可以绘制一个 OLAP 表格和图表	http://jpivot.sourceforge.net/
15	BIRT	一种基于 Eclipse 的报表系统	http://www.eclipse.org/birt/phoenix/
16	WEKA	一种基于 JAVA 环境下开源的机器学习和数据挖掘软件	http://www.cs.waikato.ac.nz/ml/weka/
17	Lucene	全文检索引擎工具包	http://lucene.apache.org/
18	Eucalyptus	一种云计算软件	http://www.eucalyptus.com/
19	Axis	一种 SOAP 引擎	http://axis.apache.org/
20	Derby	一种数据库工具	http://db.apache.org/derby/
21	Struts	运用 MVC 设计模型来开发 Web 应用	http://struts.apache.org/
22	Spring	一种降低企业应用开发的复杂性的分层框架	http://www.springsource.org/
23	Hibernate	一种开放源代码的对象关系映射框架	http://www.hibernate.org/
24	Eclipse	一个开放源代码的、基于 Java 的可扩展开发平台	http://www.eclipse.org/
25	KMail	一种电子邮件客户端	http://userbase.kde.org/KMail
26	Scilab	一种科学工程计算软件	http://www.scilab.org
27	Protégé	一种本体编辑、建模工具	http://protege.stanford.edu/
28	Jetspeed	一种采用 Java 和 XML 的开放源代码的企业信息门户的实现	http://portals.apache.org/jetspeed-2/
29	DWR	一种用于改善 Web 页面与 Java 类交互的远程服务器端 Ajax 框架	http://directwebremoting.org/dwr/index.html
30	iweb SNS	一款易于扩展的 LAMP 开源 SNS（社交网络）软件	http://www.jooyea.net/sns/index.html

总之，开源软件的分类有：第一，开发工具（如 JUnit 和 Eclipse），用于开发人员；第二，嵌入的开源软件（实际就是脚本级别的，像 Perl，JFlex，Jackyl），它们嵌在所买的产品中，对用户也是透明的，只是在服务合同中有相应的软件维护保证；第三，就是基础平台，如操作系统 Linux；第四，开源软件应用软件，如 SugarCRM、OpenOffice.org 等。

1.4 开源软件的优点

开源软件能打破其他专有软件一统天下的局面，又能让很多研发人员和商业软件公司接收，除了开源软件定义所述优点，最大的优点在于软件信息的共享，同时还具备以下优点：

1. 创新能力的分享

开源软件项目的一大特点就是创新能力的共享。从过去来看，一个开源软件项目中的很多创新可能都不是来自于原作者或者来自于一个作者，而是来自于关心这个项目的所有的人，包括商业软件公司的研发人员，他们会对项目提出一些很有见解的意见，这时，只要开源软件项目的领导者能够及时发现并采用，项目就会充满创新。而企业的动力也在于创新，当然这种创新可以来源于公司内部，也可以来源于外部。如果一个开源项目能开放的接受来自外部的创新，那么该开源项目的创新能力将被有效提高。

2. 风险均摊

由于一个开源项目可以拥有众多的自由开发者，所以软件公司的开发风险被显著降低，同时公司的投资也相应的减小，公司不必为了一个项目而雇佣大量的开发人员，特别是软件测试人员。一般公司需要雇佣几个项目的领导者，负责项目的基本协调工作，或者是主要的编程工作，而剩下的就是如何来开拓智域，让大家加入到项目中来。充分发挥开源社区的力量来降低风险，提高软件质量。

3. 软件信用提高

由于开源软件开放源代码，顾客不必担心如果公司倒闭这个软件会怎么办，他们会相对更加信任这个软件。例如，顾客一般会认为由于拥有软件的源代码，自己也是有能力修补软件的错误、甚至添加软件的新功能。

4. 软件质量的提高

一些相关开源软件研究已经显示：开放源代码软件与功能相似的商业软件相比具有更高的可靠性。由于开源软件具有更加有效的开发模式，更多的独立同行对代码和设计的双重审查，以及大部分开源软件作者对自己作品的极大荣誉感，使得开源软件项目的质量都一般相对较高。最近 Openbsd 的作者批评 Linux 内核代码的质量太差，也从侧面证明了这一点，正是由于可以查看源代码，大家可以相互比较、相互讨论、相互指正，所以软件的质量才有大幅度的提升的机会。

5. 透明度与安全性的提高

闭源软件有很多“阴暗的死角”，隐藏着许多 Bug，这些 Bug 一般需要公司的大量测试人员不断进行各种测试来找出。而对于开源项目的软件，他们的测试人员可以说是所有的软件使用者，所以 Bug 的查找工作变得相对较易。由于源代码的开放，所以安全公司的专家也可以很容易的通过查看代码的方式来找到软件的安全问题，从而立刻修补。

6. 强大的统一性

从过去的经验来看，一些开放源代码的项目是如此成功，以至于在商业方面，其竞争者无法存活；在开源设群里，没有人愿意再去与这些项目竞争，因为对一般用户来说实在太难，大家最多是在这些成功的项目中添加自己所需的附加功能。

7. 便利的宣传与推广

采用开源的项目比闭源的软件更容易获得大众的瞩目，宣传与推广也会相对的容易。也更加适合不同用户群的胃口，也更能体现开源软件的可用性。

8. 维护知识产权，拒绝盗版

对于世界上几乎每一个国家，软件盗版都是个问题。商业软件联盟（Business Software Alliance）估计盗版仅在 2002 年一年就造成了 130.8 亿美元的损失。即使像美国和欧洲这样在理论上能够负担软件费用的发达地区盗版率也分别高达 24%和 35%。而在收入较低的发展中国家，软件相对更加昂贵，盗版率可达 90%以上。随着人们意识的提高，相关法律和法规逐步健全，这种现象有逐年降低的趋势。同时，随着开源项目日趋走向产业化、成熟性，都会大大降低盗版，维护知识产权。

9. 聚集更多的优秀人才

开源社区中聚集了大量的优秀人才，他们富有激情，才华横溢，实践背景丰富，乐意为开源软件创新和奉献，如果相关组织选择了开源软件，在他们眼中，该组织充满了魅力，因此要想招揽到优秀的人才一点问题也没有。

10. 行业适应能力更强

因为开源软件大多免费的缘故，在中小型组织中迅速得到了广泛使用，这些使用开源软件的组织机构可能来各个领域，并经过长时间使用，并得到开源社区集中奉献，可以使得开源软件的适应能力更强，相比之下，闭源的软件产品通常用户数量较少，难以发现软件的缺陷，使行业适应能力不强；同时由于要付高额的费用，使得一些中小型用户感到成本高，即使软件公司承诺可以定制开发，也会是一个痛苦的合作过程。

1.5　开源软件的特点

开源软件的发展和应用，也需要从开源软件的基本特征进行分析和把握，这为进一步熟悉开源软件、理解开源软件、使用开源软件和奉献开源软件创造条件。开源软件的特点是在软件开发和使用的过程中，采用社区化和开放共享的方式，弥补了传统私有软件的公司化和封闭性的缺陷，更加适应大规模、网络化、创新型软件技术的发展需求。并在竞争中显示出低成本、高安全、易维护、促创新等优势。

1. 开源软件通常与其他开源软件混合使用

由于开源软件是由不同的个体和团队创造和奉献的，也就使得开源软件的功效专一、单一，难以处理大型、集成化的解决方案，这时就需要把相关的开源软件集成在一起进行混合使用，如典型的 SSH（Struts＋Spring＋Hibernate）集成框架。

2. 开源软件可以通过各种交互方式来解决疑惑

开源软件的持续、健康发展和应用最有效的方法之一就是通过开源社区实施。因此大多

数开源项目不仅乐于提供帮助的开发人员维护的强大的用户列表，而且还集成了 Facebook、Twitter 和 LinkedIn 作为补充的交流渠道。Bug 跟踪系统在这些项目中很常见，所以尽可能提交一个特性请求或 bug。实际上，最好的项目都欢迎这些报告，因为这可以改进项目，促进开源软件发展和应用。

3. 开源软件是开放的

开放性是开源软件最重要的特点，也符合开源软件的定义。这个开放性主要表现在通过开源社区交流，在开源软件定义下公布开发文档和源代码。

4. 开源软件更新频繁

开源软件在收集到开源社区和 BUG 相关问题后，就要进行频繁更新来提高软件的质量、可用性和健壮性，即常说的软件“迭代”，也里迭代不同于商业软件的迭代，这是因为商业软件的迭代常常对典型的普通用户是隐蔽的，难以发现软件的不足和使用范围。这是商业软件而追求经济效益所致。

5. 开源软件使用率逐年增加，但也不得不考虑影响开源软件的因素

这些问题指标主要包括产品支持的顾虑、对现有解决方案的了解、安全顾虑、缺乏支持和管理、许可或法律方面的疑虑、投资于其他供应商的结构、软件质量问题、定制顾虑、与自身的产品和服务不相关、商业供应商对于开源供应商的压力、软件成本分配策略等。

1.5.1 开源软件的成本

开源软件通过社区整合集体智慧，积累技术成果，减少重复开发，对于降低软件开发成本的作用是显而易见的。虽然开源软件有许多其他软件无法比拟的优点，但在选择、使用过程中还是不得考虑它的成本。这个成本有两方面的意思，一方面，开源软件本身的成本，另一面是开源软件作用成本。

1.5.1.1 开源软件作用成本

统计表明，开源软件的开发费用是私有软件开发模式的二十分之一。2008 年，开源软件不仅直接为软件用户节省了约 600 亿美元[①]，还间接避免了采用私有开发模式导致的项目失败成本；同样在 2008 年全球信息技术投资为 3.4 万亿美元，其中 18%～30%是基于私有软件开发模式而失败的项目[②]，数额超过 1 万亿美元。同时，通过统计数据也可以发现开源软件的成本优势和产业价值。2006 年，开源软件和服务获得 18 亿美金的收入[6]，而整个软件销售总额为 2350 亿美元[③]，按照 2008 年开源软件为用户减少 600 亿美元的成本来计算，开源软件虽然仅占有 1%的全球软件销售，却为用户节省了 25%的软件成本。另外，按照 2007 年全球信息技术投资 3 万亿的统计数据[④]，开源软件以仅占有约 0.1%的全球信息技术投资，带来了 2%的价值。

① http://news.cnet.com/8301-13846_3-9920202-62.html

② http://www.codinghorror.com/blog/archives/000588.html

③ http://siia.net/software

④ http://blogs.techrepublic.com.com/tech-news/?p=1348

开源软件的低成本是与按照许可证的要求积极加入到开源社区中开发。从社区获得的代码经过企业的独立开发和扩展，丧失了与社区代码的兼容性，不能有效发挥社区的群体智慧，然而为了维护这些代码，企业需要付出更大的成本，而且无法继续从社区代码的发展中获得新的代码，这大大增加了企业的研发成本，主要增加了代码维护成本，也容易使得代码固化，难以满足开源软件发展要求。所以，只有加入社区开发，合法使用代码，才能获得开源软件带来的低成本优势。基于此，目前很多国家政府已经认识到开源软件的低成本带来的经济效益，正在通过制定和实施鼓励开源软件的政策，大力支持开源软件基金会，开展基于开源软件的教育，并通过政府采购政策鼓励开源软件的应用。

1.5.1.2 开源软件本身成本

开源软件本身成本是指开源软件在被应用过程中所产生的成本，蔡俊杰等人在《开源软件之道》[1]中总结了部署和迁移成本、人员和培训成本、管理维护和技术支持成本和风险控制成本。

1. 部署和迁移成本

开源软件可以从 Internet 中免费获取，但在安装、配置成功运行之前，软件对用户是不能产生任何价值的。当该开源软件被使用后，它的部署和迁移成本就产生了，特别是对迁移成本，若该开源软件已长期部署了，当前需要进行迁移重新部署，这时就不得不考虑开源软件的成本，因为这一项工作需要对 Web 服务器、应用服务器、数据库服务器等进行重要部署，同时还需要重新安装相关支撑系统，并进行系统配置、初始性能调试、数据备份、数据迁移和还原及周边系统集成等。

2. 人员和培训成本

由于开源软件种类多、所涉及的范围广泛，能实现同一功能的开源软件非常多，要实现一个解决方案，可能要集成多种开源软件才能得以实现，并且实现多种开源软件配置部署也是一件麻烦的事。而目前软件人才市场上，有丰富的开源软件使用的经验的人相对较少，更多的人只会使用专用软件，或一种、几种开源软件。所以，要使开发人员快速从事研发，并非易事，需要给一定的人员进行培养。同时，为了解决这些问题，国内外相关学校和培训机构已经着手这方面的工作，甚至有些国家职业院校、大专院校都已经开设了开源软件的课程来提高对开源软件认识和掌握。

另外，开源软件由于历史原因，或多或少在可用性方面有一些不足，这也需要研发人员从技术层面和用户体验角度考虑，这种情况也为程序员带来了一些困难，直接导致这方面也需要对相关人员进行培训。因此，开源软件的人员和培训的成本也在一定程度上制约了开源软件发展。

3. 管理维护和技术支持成本

对于一般的开源软件没有技术支持服务，只能通过开源社区获得相关的技术支持，虽然目前有以开源软件定义和开源软件的许可下实现有偿的技术支持服务，但对一般的用户来说，这种方式是困难的，也难以实现的。因此，当一款开源软件或多款软件被使用了就不得不考虑开源软件在使用过程所产生的维护成本，通常意义上的维护是更新和升级，当然对于开源软件来讲，也存在更新和升级，但这种更新和升级一般是聚集开源社区智慧而实现的。一般情况下，开源软件的成本主要从易用性、安全性、升级补丁的发布频度等几个角度进行衡量。

另外，开源软件的维护也存在一定的成本，即谁来提供、解决众多用户在软件使用中遇到的问题，以及在系统出现问题、造成损失，谁来负责等一系列问题，对一般的用户来说，他们首先想到的就是找开源社区。

总之，软件的维护和技术支持费用取决于具体的开源软件和用户群，从某种意义上讲，只要某开源软件使用的用户群多，则在开源社区讨论的就多，相应也更能发现一些缺陷，从而提高软件的更新和升级的频度。

4．风险控制成本

选择并使用开源软件存在巨大的风险成本，主要表现开源软件的潜在的知识产权纠纷，由于在市场上开源许可证众多，要使开源软件在这些许可证下想到兼容，相互不抵触并非易事。这是因为每一种许可证条款有差异，就需要根据具体的应用情况做具体分析。同时，任何人都可以在开源社区提交代码，这样难确认源代码的来源及合法性。

另外，开源软件在使用过程所存在的风险，如是否能导致基于开源软件的系统崩溃，开源软件到底安不安全等问题，因此，开源件还在一定程度上存在健壮性、安全性等风险。

为了有效控制开源软件应用的风险，在选用开源软件时，需要建立严格的评估流程，以确保所选的开源软件是成熟可靠的。

1.5.2　开源软件的成熟度测评概况

一般认为，软件的成熟度是通过评估和测评来确定的，当然开源软件也不例外。因为开源软件自身的特点，决定了开源软件在成熟度评估中，软件设计评价和代码评测成为整个评估过程中的重要一环。商业软件的代码封闭特点使得评测人不能对代码质量进行量化测试。这也是开源软件之所以更适合与精确评测的一个重要条件，更能比商业软件容易发现软件的缺陷，从而改进软件的质量，提高软件的成熟度。

1.5.2.1　开源软件成熟度评测简介

软件缺陷是模式与测试是 2000 年后在美国诞生的一种新型软件测试技术，是指与需求不一致的统称为软件缺陷，主要分为功能性与非功能性（能给出确切语法和语义定义，并在实践中经常发生或后果比较严重的缺陷的集合）两种缺陷，缺陷发生机理是由疏忽、二义性、不理解、遗漏四个方面构成，并由故障、安全漏洞、疑问代码、规则四个类型组成，如图 1-4 所示。它是国际目前最流行的软件测试技术之一，是可信软件系统中必须要做的一步测试，是美国政府指定众多大企业都普遍使用的测试技术，并主要以软件中的非功能性缺陷，以缺陷检测效率高、定位准确、自动化程度、易用等特点而受到广泛关注。目前常采用 Klocwork 代码静态分析工具和软件架构分析工具对项目的程序和设计进行量化分析，进行定性评估。

而 Klocwork 公司是软件静态分析领域技术和市场领先的厂商，全球拥有 200 多个客户，其中许多是全球财富 500 强中的公司。Klocwork 软件是 Klocwork 公司基于专利技术分析引擎开发的，综合应用了多种近年来最先进的静态分析技术，是出色的软件静态分析软件。Klocwork 产品与其他同类产品相比，具备缺陷检测安全漏洞检测、软件架构分析、软件度量分析、可定制的代码分析、开发人员 IDE 集成，且具有很多突出的特征：

（1）Klocwork 支持的语言种类多，能够分析 C、C++和 Java 代码；

（2）能够发现的软件缺陷种类包括软件质量缺陷，又包括安全漏洞方面的缺陷，还可以分析对软件架构、编程规则的违反情况；

（3）软件分析功能既能分析软件的缺陷，又能进行可视化的架构分析、优化；能够分析软件的各种度量；

（4）能够提供与多种主流 IDE 开发环境的集成；能够分析超大型软件（上千万代码行）。

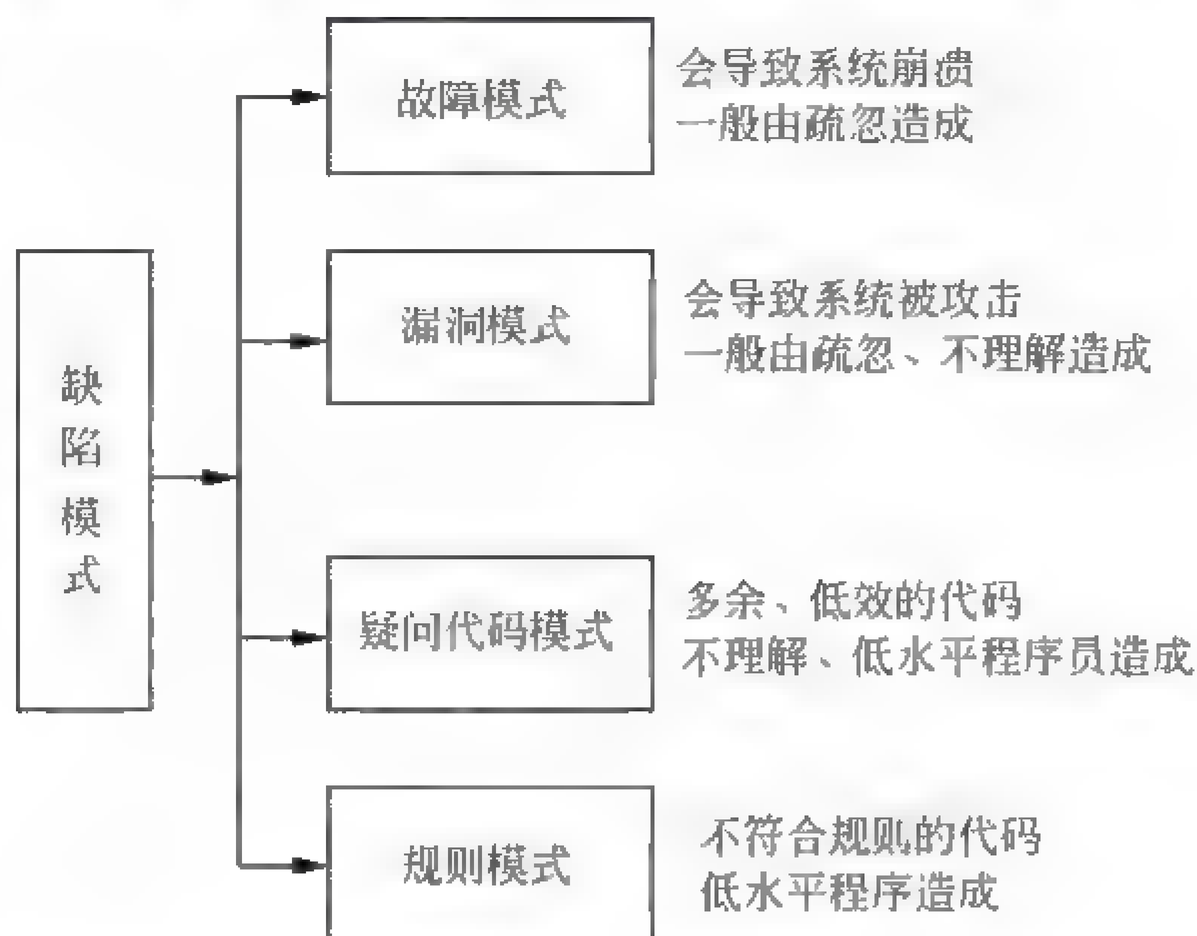

图 1-4　软件缺陷模式（出自北京邮电大学网络与交换技术国家重点实验室）

1.5.2.2　开源软件的成熟度模型

相关开源软件的成熟度模型的建立，国外有 QualiPSo 的研究成果，NEAOSS（Northeast Asia Open Source Software），以及以中国开源社区为平台的 CSIP。

1．QualiPSo 简介

QualiPSo 项目的目标之一就是开发出一个类似 CMMI（Capability Maturity Model Integration）的开源软件成熟度度量模型。此模型致力于提高开源软件开发过程及其产品的可信任度。并根据 CMMI 可信任度的相关定义，映射出在开源软件领域中开源成熟度模型（OpenSource Maturity Model，OMM），如图 1-5 所示。同时在参考 CMMI 模型的同时，又除去其复杂而不利于操作的部分。因此，OMM 模型虽然采取了类似于 CMMI 的分层机制，但是在内容方面又有所不同，它更容易理解，并为大多数人所接受[①]。

2．NEAOSS

NEAOSS 论坛由中国、韩国和日本三国的政府主管部门和致力于推进开源软件的区域组织发起，这些区域组织包括中国开源软件推进联盟、韩国开源软件推进论坛和日本开源软件推进论坛。论坛的目标是在东北亚地区促进开源软件的发展。NEAOSS 论坛于 2004 年 7 月成立了“第三工作组：标准和认证研究”（WG3） 以开展开源软件相关标准化和产品认证方面的研究，其中就包括开源软件成熟度。WG3 于 2010 年 7 月 14-15 日在北京召开工作组第十三次会议，讨论《开源软件评估》项目并提出工作计划，成立 OMATF（Open Source Software Assessment）开源软件评估任务组，其主要内容包括建立开源软件分类、建立开源软件评估

① http://www.qualipso.org/sites/default/files/A6.D1.6.3CMM-LIKEMODELFOROSS.pdf

框架、挑选待评估的开源软件列表、收集开源软件具体信息并进行评价。

并在 2010 年，中日韩三国代表在多次研究协商后达成一致，制定了开源软件成熟度评估细则，并分工对 300 多个开源项目软件进行信息核查与数据评估。目前，开源项目已经成功审核完成。2011 年，将继续对第二批 100 余个开源项目进行评估。对于开源软件的成熟度，可以分为 License/Develop System/Version/Tools/Specification/Composition/Quality/ Recognizability/ Package/Business/Case 11 大类[①]。

3．CSIP

CSIP 主要提供开源软件测评收费服务，为提交申请的客户提供测评任务，主要业务包括开源软件成熟度选型和商业/开源软件比对测试两种，它的业务流程如图 1-6 所示。它是以开源软件长期研究成果和开源商业应用丰富信息为基础，并建立了开源软件成熟度评估规范，并基于中国开源黄页网（http://yp.oss.org.cn ）的丰富资源，通过软件质量评测、量化质量属性、成熟度模型计算等一系列评估方法，客观地评价开源软件的成熟度，并形成评估报告，从而帮助中小企业及开源软件集成商成功部署开源软件商业应用，并帮助不断提高开源软件质量。

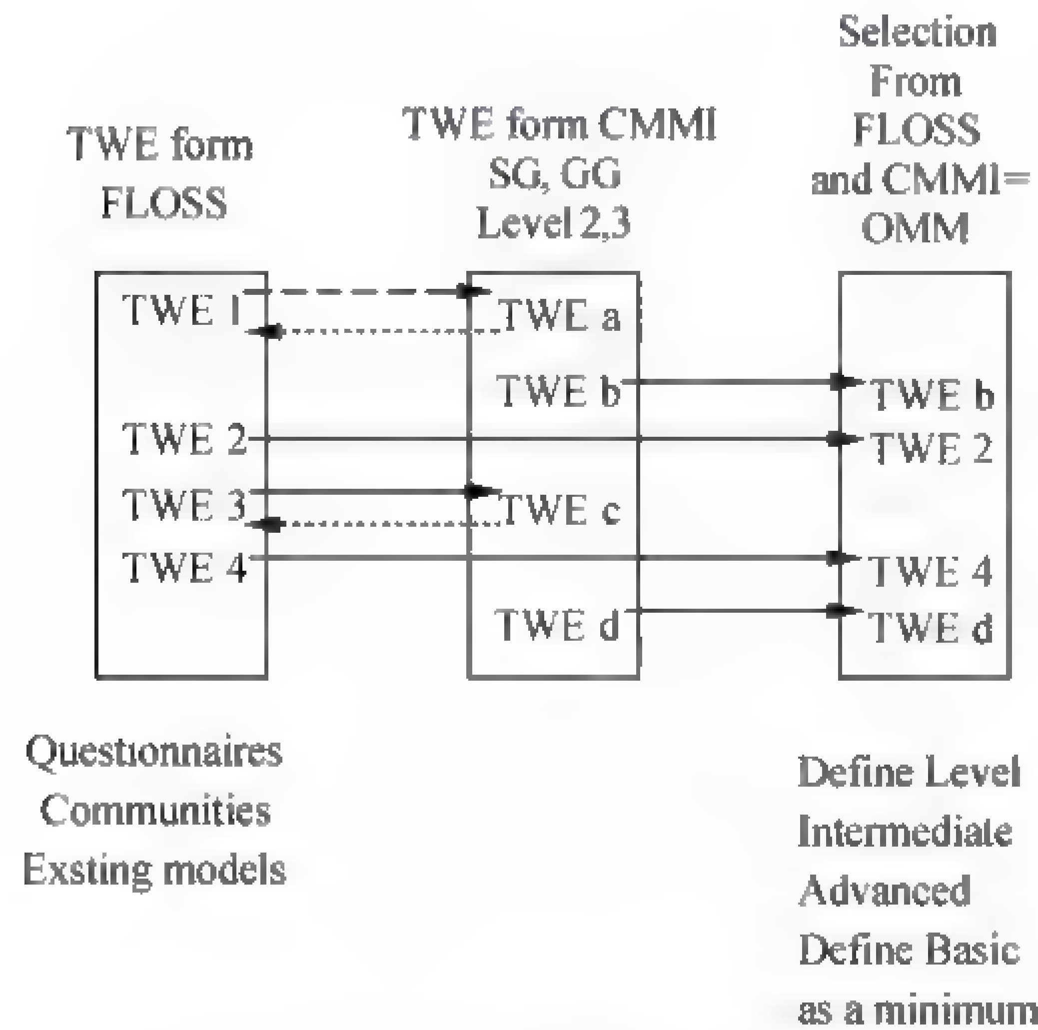

图 1-5 CMMI 映射出 OMM

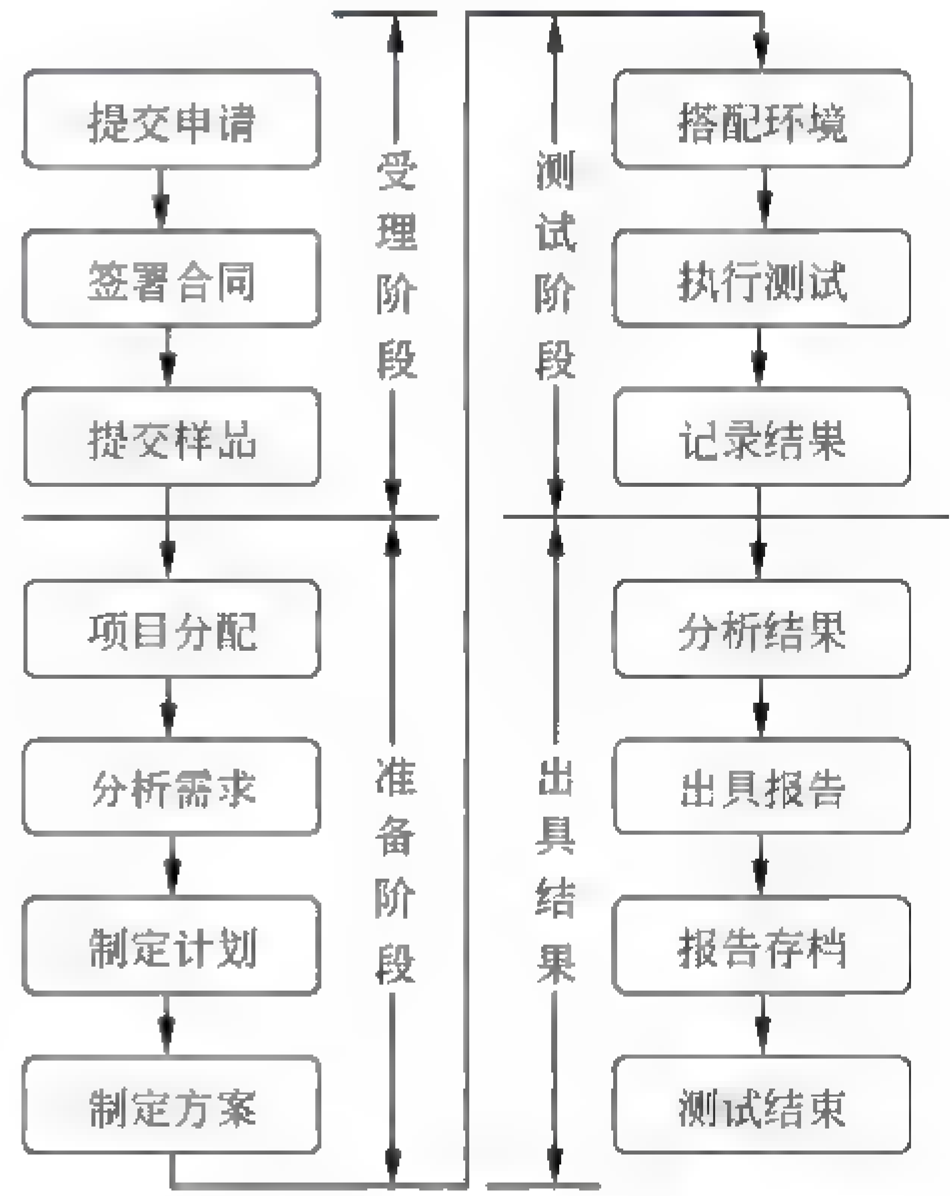

图 1-6 CSIP 开源软件测试业务流程
（来源：http://www.csip.org.cn/yewu/48.html）

1.5.3 开源软件的选择策略概述

开源软件越来越主流化，使用范围也越来越广。但企业需要确保自己所使用的开源软件具备可信赖性，而且基于开源软件开发的软件系统所具有健壮性如何，等等，这些都是在选

① http://www.fsstd.org.cn/getIndex.req?action=quary&req=modulenvpromote&id=1568&type=0&moduleId=896&sid=43

择开源软件时所注意的现实问题。于是，开源许可证变得越来越重要，也越来越多和复杂，使其实现起来并不容易。据报道，如今业内有超过三十多万个开源软件供各种类型的企业和个人使用。这样如此繁多的开源软件，不仅要保证其开源许可证，还要保证其高可靠性和健壮性，这对于使用者来说，无疑是个难题。要选择正确的开源软件，最基本的判定方式就是通过开源许可证，目前，也有相应的权威部门和机构来进行开源软件的认证，即对开源软件的成熟进行测评，为选择开源软件提供参考和帮助，如 1.5.2 小节中的 QualiPSo 开源软件成熟度模型，NEAOSS 论坛和 CSIP 开源软件测评机构，这样就可以得到一套切实可行的认证管理以及说明文件供企业和开发人员参与。以下选择开源软件选择的具体策略：

1. 缩小开源软件选择范围

根据开源软件的分类缩小开源软件的选择范围，并通过开发需求设置选择条件，然后在所在的开源软件分类中按条件选择。这种方式简单，易操作，但当某一类开源软件数目过多，这种方式就会显得力不从心，并且即便是选择好了开源软件也难以保证该开源软件的许可证完备、可靠性高和健壮性好等要求。有一点儿可以保证在进行开源软件分类选择时，根据企业和开发者的经验是可以选择到满足需求的开源软件的，但 SSH 框架、Java SOS（是由一组用于快速建站可配置的 Java servlets 所组成。包括 Forums、Chat、Calendar、HttpProxy 等 servlets）、LAMP 等者是得到了开源社区、开发者、企业认可的开源软件除外。

2. 开源软件测评方法

通过权威的开源软件测评机构对相关的开源软件进行测评，从而选择一种适合需求的开源软件，但这一种选择方式可能要花费一定的测评费，但可以得到一份详细的测评报告供用户决策。这一种方式的前提必须先选择一些开源软件，再找测评机构测评，其实这个过程是很复杂的，即怎样从诸多的开源软件列表找出按需求的开源软件。这种方式可以对较熟悉的开源软件中使用。

3. 开源软件的活跃度

一种活跃度强的开源软件其生命力就强，相应的使用人数也会很多，在开源社区交流也就多，因此，选择活跃度高的开源软件作为需求是能可以降低软件开发的风险的，特别的选择在一段时间内有稳定的活跃度的开源软件能回避掉一定的风险。

4. 开源软件伙伴生态环境

全面考虑所选择的开源软件是否具备好的可扩展性，可操作性，且能与所支持的基础平台是否兼容，即开源软件伙伴生态环境是否和谐。同时，这个开源的伙伴生态环境是否满足开源软件的许可证，并能与各开源社区实现有效的互动，以达到提高使开源软件所带来的经济效益。

5. 成功案例

以成功案例来衡量开源软件选择标准，毕竟成功案例越多，就可以直观表明该开源软件可用性、健壮性好，也能为客户增强信心，并能清楚了解到哪一款开源软件的优劣势。因此，选择成功案例多的开源软件作为项目开发是一种较直观的方法。

6. 开源软件后续发展蓝图

一款开源软件的成熟状况一般是由开源区的更新频率、反馈信息、参与人数等来决定，这些参数为后续的发展蓝图提供具体的指标，使其逐渐成为一款成熟、缺陷少、普通能让用户接受的软件。因此，在选择开源软件时，就可以从后续的发展蓝图来确定软件选择。

7. 开源软件的互操作性

所谓互操作性（Interoperability）是指一个软件系统与另一个软件系统互相间具有接收、处理并共享所发送信息的能力。而兼容性也可以归纳在互操作性的范畴内，所谓兼容性（Compatibility）指某个系统上运行的应用程序符合另一个系统的接口要求，从而使该应用程序也可在另一个系统上运行，这时对该应用程序符合某个接口的能力称为兼容性。

各种开源软件与各种私有商业软件在各自相互之间的互操作性问题是近一段时期以来，开源软件必须关注的问题，就目前来开源软件使用情况显示，很多还是与商业私有软件共存使用，这就决定在选择开源软件时需要考虑其互操作性。同时，在与商业私有软件共存使用时必须考虑开源软件的协同工作能力和软件可信水平。

1.5.4 开源软件的管理机制

开源软件的管理机制是指在开源软件的许可证和定义下，实现开源软件管理的一种模式。开源软件自诞生以来，就是在许可证范围内实现开源软件管理和商业化进程，但由于开源软件的数量巨大，种类复杂，开源社区多样，这对开源软件的有效管理带了诸多的困难。毕竟建立完善的开源软件管理体制可以规范企业对开源软件的使用，提高开源软件的使用效率，从而有效控制开源软件所来的风险。

开源软件的具体的管理机构可以根据开源软件的应用策略制定开源软件的管理流程、管理目标、管理过程，并建立相应的组织结构、开源软件评估中心、决策机构、审计机制、开源社区激励机制、员工培训和实训中心等，如图1-7所示，从而提高开源软件价值。

对用户来说，价值主要体现在：

（1）开源软件所具有的多样性、可替代性、自由性和共享性。

（2）降低使用成本，且更能受到其他用户的关注。

（3）防止供应商锁定，使用户获得更大的自由性和自主性。

（4）提高用户使用的透明度性和安全性，减少软件漏洞带来的风险。

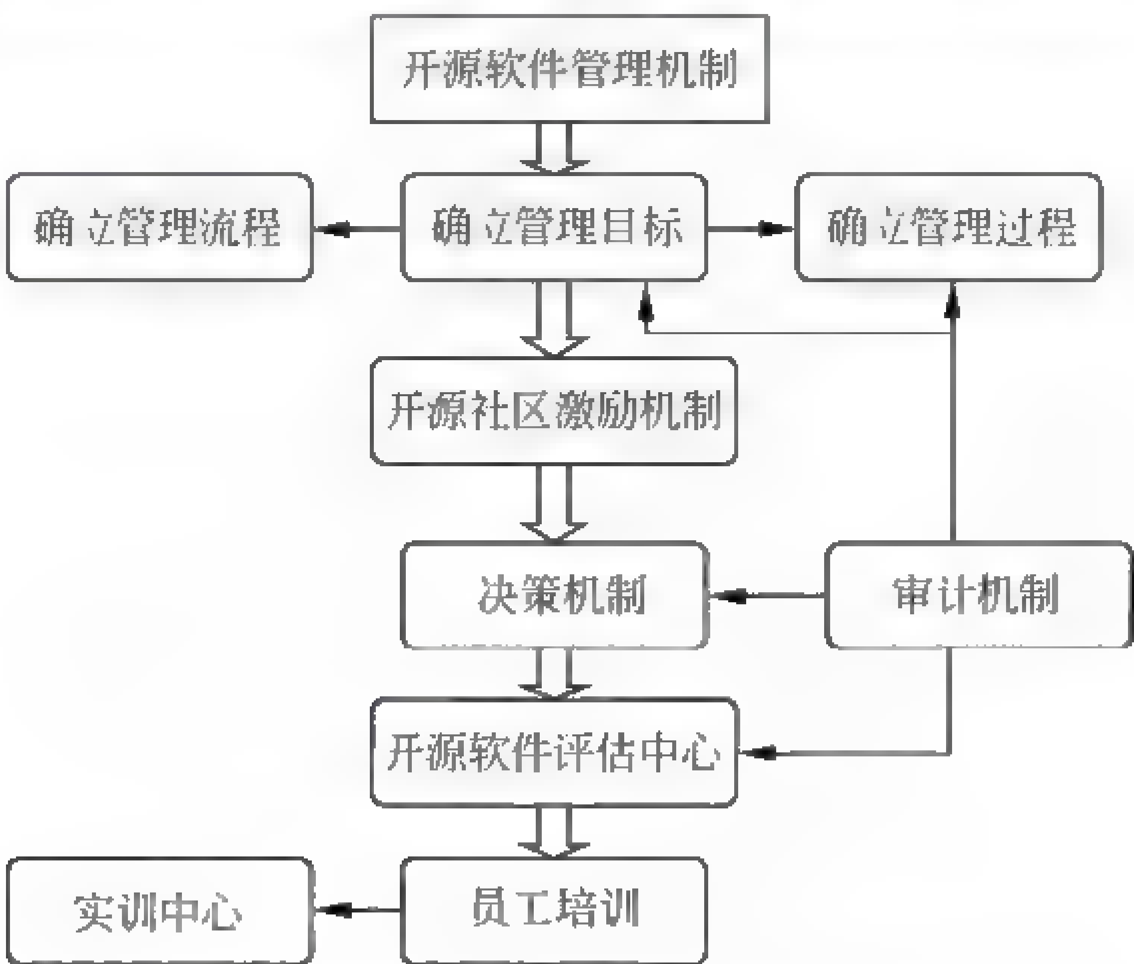

图1-7 开源软件管理机制结构与流程图

对于研发人员来说，价值主要体现在：

（1）提高更快的研发速度，以获得最大的价值。

（2）获得更低的开发成本，为企业创造更多的商业价值。

（3）减小研发人员的重要劳动，提高研发人员的开发效率。

（4）以最快的速度获得修正错误，以便及时更新软件。

1.5.5　开源软件与知识产权

开源软件并非完全免费、无知识产权、无法律责任，它仍有著作权、软件许可证和专利等，因此，相应地受到了著作权法、合同法、专利法的保护，对用户使用开源软件实行了一定的约束，特别是商业应用更是要注意，若不处理好开源软件的知识产权问题，就有可能违反相关法律并受到法律的制裁，即主要表现形式就是巨额赔偿。其实开放源代码软件就是在 GNU 通用公共许可证（GPL）下发布的软件，以保障软件用户自由使用及接触源代码的权利。这同时也保障了用户自行修改、复制以及再分发的权利。简而言之：所有公布软件源代码的程序，都可以称为开放源代码软件。如表 1-3 是开源软件与其他类型软件传播性对比。

表 1-3　开源软件与其他型软件传播性对比

软件名称	是否免费	是否允许复制	是否允许分发	是否允许修改
开源软件	不一定	是	是	是
自由软件	是	是	是	是
商业软件	否	否	否	否
试用软件	试用期内免费	是	是	否

开源软件的著作权理论上属于原创软件作品的作者（writers、authors、developers），以及升级软件作品的后续修改者（贡献者 Contributors，志愿者 Volunteers），总称为所有者（owners）。它实际上是有条件地将某些著作权的权能无偿转移给了所有愿意接受其采用的许可协议的人，实质上作者只是放弃了部分著作权（修改权、保护作品完整权以及复制权），并将这些放弃了的权利有条件地授予了那些愿意接受此条件的人。但开源软件的著作权人依然享有著作权法所赋予他的其他权能（如署名权等）和处置这些权利的自由[7]。

早在 2004 年，美联储、美国联邦存款保险公司和其他联邦金融管理机构在共同发布的“免费开源软件风险管理”指南中简要描绘了使用开源软件的各种策略和法律风险，同时，开源软件一旦产生就具备了著作权，加之信息领域的私有标准（在市场上占有垄断地位的软件厂商所使用软件自然形成的一种规范）所引发专利收费等一系列知识产权问题，也导致出现了较多的此类案件，如在 2002 年，MySQL AB 控告 Progress Nusphere 公司违反使用 GPL 许可证所要求的使用条件发布派生产品，最终庭外和解；2003 年，美国 SCO 公司控告 IBM 将其 Unix 操作系统中 UnixWare 的 V 程序代码移植到 Linux，要求 IBM 就不正当竞争、违反合同和侵犯商业秘密给予 10 亿美元的赔偿；2004 年，Netfilter/iptables 项目控告 Sitecom Germany 公司违反 GPL 许可证，最终，Netfilter/iptables 因此获胜；2006 年，德国 GmbH 的 D-Link 德国分部因违法 GPL 受到了惩罚；2007 年，BusyBox 公司和软件自由法律中心（Software Freedom Law Center）控告 Monsoon Multimedia 等公司违反了 GPL 许可证的提出的要求，最终，以修改软件和做出相应赔偿；同样在 2007 年，自由软件基金会控告 Skype 基于 Linux

的Skype WiFi电话使用了GPLv2代码，但却没有按照许可证的要求发布修改后的代码，最终，Skype败诉；2008年，自由软件基金会控告Cisco的Linksys品牌下的多个产品使用了GPL许可证协议，但未遵守GPL协议的要求，最终使得在2009年和解并向自由软件基金会提供资金支持；在2009年9月在巴黎AFPA上诉法院裁决EDU4公司违反了GNU GPL协议，在分发软件时只提供了二进制文件，而拒绝提供源代码；在2010年初，美国加州一家名为Cybersitter的软件公司向当地法院提起诉讼，称"绿坝一花季护航"互联网色情信息屏弊软件抄袭了它的逾3000行代码，并索赔22亿美元。

综上述，开源软件的许可对用户限制最少，对权利人限制最多，这对讨论开源软件的知识产权是必要的，也是减少商业应用中出现的商业纠纷。而商业软件最大的成功就是依赖于知识产权，包括版权保护、营销策略、捆绑许可、专利保护和商标保护。自由软件和开源软件一旦商业化，一定也会依赖和利用知识产权，如果不利用知识产权商业化就不会取得成功。知识产权有利于防止垄断，能够促进创新。

开源软件作为一种新兴的软件模式，与传统的商业软件相比具有其自身的特点，主要从著作权、专利权、商标权、商业秘密等四个方面来体现开源软件知识产权的形式。

1.5.5.1 著作权

根据《中华人民共和国著作权法实施条例》第二条规定："著作权法所称作品，指文学、艺术和科学领域内，具有独创性并能以某种有形形式复制的智力创作成果" 。《计算机软件保护条例》也指出：计算机软件是受著作权法保护的一种作品类型。同时，《计算机软件著作权登记办法》第二条规定，"为促进我国软件产业发展，增强我国信息产业的创新能力和竞争能力，国家著作权行政管理部门鼓励软件登记，并对登记的软件予以重点保护。"《计算机软件著作权登记办法》第七条也规定，"申请登记的软件应是独立开发的，或者经原著作权人许可对原有软件修改后形成的在功能或者性能方面有重要改进的软件。"

GPLv2的序言中提出："大多数许可证剥夺了大家共享和修改软件的自由，相比之下，GPL则力求保证共享和修改软件的自由"。为了实现这一目标，GPLv2采取两项措施来保护你的权利：给软件以著作权保护和提供法律许可。为了保证用户复制、发布和修改这些软件；GPLv2"有关复制、发布和修改的条款和条件"部分中规定，"此许可证适用于任何包含版权所有者声明的程序和其他作品，版权所有者在声明中明确说明程序和作品可以在GPL条款下发布。

2007年6月发布的GPLv3版本在导言也指出[1]：大多数软件及其他具有实用价值的作品之授权是设计用以剥夺共享和修改该作品的自由。相反地，GNU通用公共授权力图保证分享和修改一个程序的所有版本之。GNU通用公共授权的设计是为了确保拥有发布自由的软件副本、以及为此收费的自由，确保研发者能收到原始码或者在需要时能获取原始码，确保能修改软件或者将它的一部分用于新的自由软件当中，并且确保能做这些事情。为了保障你的权利，我们需要禁止任何人否定你拥有这些权利，或者要求你放弃这些权利。因此，当你

① http://wiki.linux.org.hk/w/GPLv3

散布此软件的副本或者修改它的时候，有些责任你必需肩负，那就是，尊重他人的自由。GPLv3 使用 GNU 通用公共授权的开发者通过两项措施来保护的权利：声明软体的版权，以及提供本授权文件，以赋予复制、散布及/或修改软件的法律许可。

与 GPLv2 相比，主要有以下四条改动：

（1）解决软件专利问题；

（2）与其他许可协议的兼容性，如 MIT/X 许可协议、BSD 许可协议、LGPL，都是与 GPL 兼容的；

（3）源代码分区和组成的定义；

（4）解决数位版权管理（DRM）问题。

同时，著作权授予软件的权利包括人身权和财产权两部分：

（1）人身权包括署名权、发表权、保护作品完整权和修改权；

（2）财产权主要包括复制权、发行权、出租权、汇编权和翻译权。

由此可见，GPL 并没有放弃软件的著作权保护，而是以现有著作权制度作为基础，通过 GPL 等软件著作权许可协议实现共享和自由的理念。开源软件是享有著作权的作品，这在法律和司法实践中均予以了认可，因此也就具有了进行著作权登记的资格。这为进一步合理、正常使用开源软件提出了法律依据。而且大部分开源软件都受到著作权保护，就决定用户要在相关法规和开源软件许可证下正确使用开源软件，避免造成对作者的侵权，造成不必要的纠纷和巨额赔偿。同时，开源软件著作权约束和规定也很麻烦，这是由于当前供开源软件采用的许可证较多，参见表 1-4 所示[8]，部分许可证兼容性关系如表 1-5 所示[1]，在表中概述了每一个许可证的特征。如此复杂的许可证容易把风险转移到用户身上，这是由于要检查如此多的开源软件代码的来源，并能验证其著作权的合法性是一件比较困难的问题，而且开源软件代码来源广泛，主要有程序爱好者、自由职业者、商业公司捐赠、其他软件项目中复制派生出来的代码和商业公司指派公司员工参与所贡献的源代码。这些因素决定了开源软件的著作权具备多样性和难预测性等不确定性。

表 1-4　常见开源软件许可证特征对比

项目	是否允许可以同其他非开放源代码软件代码混合	是否可以将对源代码的修改不公开	是否明确了专利许可授权	是否明确了专利侵权诉讼导致许可证协议终止	是否明确禁止与函数库连接	是否只能按本许可证发布源代码	是否要求对于获得的源代码可能存在的知识产权进行以“LEGAL”为抬头的提示
GPL 许可证	否	否	否	否	是	是	否
LGPL 许可证	是	否	否	否	否	否	否
BSD 许可证	是	是	否	否	否	否	否
NPL 许可证	是	是	否	否	否	否	否
MPL 许可证	是	是	否	否	否	否	否
APACHE 许可证	是	是	否	否	否	否	否
QPL 许可证	是	是	否	否	否	否	否
QNCL 许可证	否	是	否	否	否	否	否
Ricoh 许可证	是	是	是	是	是	不一定	是

续表

项目	是否允许可以同其他非开放源代码软件代码混合	是否可以将对源代码的修改不公开	是否明确了专利许可授权	是否明确了专利侵权诉讼导致许可证协议终止	是否明确禁止与函数库连接	是否只能按本许可证发布源代码	是否要求对于获得的源代码可能存在的知识产权进行以“LEGAL”为抬头的提示
SISSL 许可证	是	否	是	是	否	不一定	否
SPL 许可证	是	是	否	否	否	否	是
Jabber 许可证	是	是	否	是	否	否	是
MOTOSOTO 许可证	是	是	否	是	否	否	是
NOKOS 许可证	是	是	是	是	否	不一定	是
Open Group Test Suite 许可证	是	是	否	否	否	是	是
AFL 许可证	是	是	是	是	否	否	否
Artistic 许可证	是	是	是	是	否	否	否
APSL 许可证	是	是	是	是	否	否	否
Common 许可证	是	是	是	是	否	否	否
IBM 许可证	是	是	是	是	否	否	否

表 1-5　部分许可证兼容性关系

	MIT	新版 BSD	Apachev2.0	MPL v1.1	LGPL v2.1	LGPL v3.0	GPL v2	GPL v3
MIT	是	是	是	是	是	是	是	是
新版 BSD	否	是	是	是	是	是	是	是
Apache v2.0	否	否	是	是	否	是	否	是
MPL v1.1	否	否	否	是	否	否	否	否
LGPL v2.1	否	否	否	否	是	否	是	否
LGPL v3.0	否	否	否	否	否	是	否	是
GPL v2	否	否	否	否	否	否	是	否
GPL v3	否	否	否	否	否	否	否	是

1.5.5.2　专利权

著作权仅仅保护计算机软件创作的表达，无法保障创作背后的概念和操作使用的方法，更无法保护其与硬件结合而产生的新技术，而专利能够保护其具体的方法和功能，而且专利享有绝对的排他权，在专利有效期内，权利人能够禁止他人实施专利技术，即使该技术是他人独立开发的也不例外，因此商业软件一直都在积极寻求软件专利保护[7]。而自由开源软件在版权包括许可协议保护方面取得了较为宽松的环境，对个人要求较宽，对商业应用较严，但仍然躲不开专利的追索。但用户一旦侵犯专利权，不但要追溯开源软件“发行者”的法律责任，还要追溯“使用者”用户的法律责任，这与著作权有明显的区别。同时，开源软件许多程序是由全球志愿者集体编写、合作开发的，其中难免携带“隐性专利”。为了改变“GPL许可证只能解决版权问题，不能解决专利问题，特别是隐性专利” 的现状。介于此，自由软件基金会首席律师Eben Moglen领导制定了GPLv3，试图改变GPL许可证只能解决版权问题，不能解决专利问题的现状，进一步提升遵守 GPL 的开源软件能具有专利的权利。特别是 2007年 5 月微软声称开源软件侵犯其 235 项专利权，导致开源软件知识产权问题进一步引起业界

高度关注。开源软件获得著作权与专利的双重保护，引起了开源软件业界的广泛关注。许多开源软件的支持者，包括Linux Torvalds本人也明确表示反对软件专利制度，因为他们认为，软件专利化趋势会给开源软件的发展带来巨大冲击。不利开源软件在自由、共享的环境中继续进步，但相关企业、机构为了追求商业利益，为了保护自己的智慧成果不得不拿起专利的法律约束来保护自己的成果，从另一角度也可以发现，真正的要使开源软件继续发展，少被专利束缚，最好的办法就是大家一起来奉献，改变多索取不奉献的现状。使得软件专利的独占性与开源软件所倡导的“自由共享”精神相反，软件专利化目前已经成为商业软件对开源软件的最大威胁。可以预测，在不久的将来，开源软件的自由共享将逐渐被商业化颠覆，如先有自由软件，但自由软件太自由，基本不符合商业化的要求，让自由软件发展举步为坚，最终出现了满足商业化要求的开源软件，但近来就逐渐将开源软件的一些方法专利化，越使开源软件具有较重的商业气味，最终使公开的源代码成为“死”代码，让一般的用户不敢触摸，这样何以让已公开的源代码得到更快的更新和升级？因此，实行开源软件的专利化是一个非常难处理的问题。

由于开源软件专利化，也会增加更多的商业纠纷。但也有观点指出：一方面专利保护能更强有力地促进软件技术的创新。另一方面，专利对后续软件开发者的限制也更大，特别是开源软件的开发人员试图避开专利开发同类软件的难度明显提高。使得在开源软件程序原始开发或后继修改的过程中，若后继贡献者在程序的修改或演绎作品中使用了他人享有专利的某项技术，则很有可能面临专利侵权的诉讼。如2006年Red Hat在收购开源软件公司JBoss几个星期之后就遭到了与JBoss有关的专利诉讼。FireStar软件公司向得克萨斯州美国地区法院提出起诉，指控JBoss Hibernate 3.0软件侵犯了其连接关系数据库与面向对象的软件的技术专利。因此，一旦开源软件专利化，相关的纠纷也将继续延续下去，这对开源软件快速发展带来一定程度上的考验。又由于专利具有地域性，作为一个基本原则，各国专利制度独立，即使是专利国际条约的缔约国，他们各自的专利制度也是存在差异的。但开源则是一项国际性的运动，几乎没有一个开源软件仅局限在一个国家或地区发展。专利制度的地域性和开源运动的国际化，导致开源软件专利风险的产生。目前，各国对软件专利的态度和审查标准均存在较大差异[7]。这给开源软件的方法专利化带来了极大的风险和不可控性。

1.5.5.3 商标权

软件商标是指软件生产者为使自己开发、制造的软件区别于其他软件产品而置于软件包装表面或软件运行时屏幕中所显示的文字、图形等特殊标志。开源软件的商标同其他软件产品一样，同样受到商标法的保护。Linux Torvalds在1991年发布了程序内核后，并没有将Linux或LINUX注册为商标。数年之后，美国人William R.Della Croce在美国抢注了LINUX商标，使用商业类别为计算机类产品，之后，他向《LINUX杂志》和其他一些以LINUX产品为主的商业软件公司致函，要求他们支付商标许可证使用费。

这一诉讼震惊了整个开源软件业。直到1997年，诉讼双方才正式达成庭外和解，并将注册商标所有权移给公认为程序内核开发者Linux Torvalds。虽然这样的诉讼看上去不算道德，但从法律层面上看就是正当的。Linux Torvalds或许吸取了美国抢注的教训，他随即在其他国家着手进行商标注册申请。但在2005年，澳大利亚知识产权局却以四条理由驳回了其就LINUX要求注册商标的申请，这个案例再一次给开源软件业的商标使用政策敲响了警钟：要

想开源软件产业能够健康地发展，仅仅依靠获得了某个商标是不够的，还需要商标的所有权人合理地、正确地使用商标[9]。有人曾向美国专利商标局申请把“Open Source”作为商标，但未被批准。因此，开源软件的商标权主要有商标叙述性和指标性使用。

1.5.5.4 商业秘密

除了版权法、专利法、商标法之外，开源软件涉及的知识产权问题还包括合同法、反不正当竞争法等多部法律，这些法律从各个方面实现了对软件的全面保护。特别是商业秘密的保护，即在软件开发过程中凝聚了许多人员的大量创造性的智力成果和开拓性创新思维，但大部分往往不能得到著作权或权专利保护，使得这些成果可以被其他人应用。若要保护这些思想内涵，只能借助于商业秘密保护。但是由于开源软件宗旨就是实现源代码开源，要求源代码和目标代码全部公开，这就排除了商业秘密保护的适用方式。

开源软件的全部技术是由以开放源代码所表征的公开的技术与不公开的工程化实现技术两部分组成。依据我国《反不正当竞争法》第十条提出商业秘密是指不为公众所知悉，能为权利人带来经济利益，具有实用性并经权利人采取保密措施的技术信息和经营信息”。而为了更好地保护开源软件这一优秀的软件模式，提高开源企业服务市场的竞争能力，促进开源软件发展，使更多的人员能奉献自己的潜能，以减少诉讼成本，可以采取以下具体措施[7]来保护开源软件在商业应用中的秘密：

（1）加强开源软件开发过程中文档的管理。

（2）开源软件企业应限定软件开发人员的保密范围。

（3）重视诉讼前的证据保全工作。

（4）在诉讼中应当提交的齐全的证据。

1.6 最有价值的开源软件

在2009年，美国知名科技网站-信息世界（Infoworld.com）评出了近年来最有价值10款开源软件。

1．Linux内核

Linux内核是由C语言写成，符合POSIX标准的类Unix操作系统。在1991年，Linux是由芬兰黑客Linus Torvalds为尝试在英特尔x86架构上提供自由免费的类Unix操作系统而开发的，他最初禁止将Linux进行任何商业化，但后来他改用GNU通用公共许可证第二版。该协议允许任何人对软件进行修改或发行，包括商业行为，只要其遵守该协议，所有基于Linux的软件也必须以该协议的形式发表，并提供源代码。现在Linux已成为最受欢迎的自由电脑操作系统内核。

2．GNU工具及编辑器

GNU工程成立于1984年，旨在开发一个完整的类似于Unix的操作系统，它所倡导的自由软件给开发者带来了福音。并且是完全免费的完整操作系统和配套工具，遵循GNU通用公共许可证（GPL）协议，任何人都可以从网上获取全部的源代码。

3. Ubuntu

Ubuntu 是一个由社区开发的基于 Linux 的操作系统，是一个由全球化的专业开发团队建造的操作系统，它包含了很多应用程序，如浏览器、Office 套件、多媒体程序、即时消息等。它作为一个基于 GNU/Linux 的平台，Ubuntu 操作系统将 ubuntu 精神带到了软件世界。且 Ubuntu 项目完全遵从开源软件开发的原则；并且鼓励人们使用、完善并传播开源软件。也就是说 Ubuntu 目前是并将永远是免费的，由 Canonical 公司以及全球数百个公司来提供商业支持，包含了由自由软件团体提供的最佳翻译和人性化架构等承诺。

4. 三款 BSD 操作系统（FreeBSD、NetBSD 和 OpenBSD）

BSD（Berkeley Software Distribution，伯克利软件套件）是 Unix 的衍生系统，即是由经过 BSD、386BSD 和 4.4BSD 发展而来的类 Unix 的一个重要分支。在 1977—1995 年间由加州大学伯克利分校开发和发布的。它支持 x86 兼容（包括 Pentium® 和 Athlon™）、AMD64 兼容（包括 Opteron™、Athlon™64 和 EM64T）、ARM、IA-64、PC-98 以及 UltraSPARC® 架构的计算机。并且在 20 世纪 80 年代，BSD 广泛的被工作站级别的厂商所接受，并且衍生出了许多变形的 UNIX 授权软件，如比较著名的例子如 DEC 的 Ultrix，以及 Sun 公司的 SunOS。而且具备先进特性、强大的互联网解决方案、能够运行大量可供选择的应用、易于安装等优点。目前在国内有很大市场的是 FreeBSD。

5. Samba（允许 Linux 和 Unix 服务器为 Windows 客户端提供文件和打印服务）

Samba 是一种自由软件，用来让 UNIX 系列的操作系统与微软 Windows 操作系统的 SMB/CIFS（Server Message Block/Common Internet File System）网络协定做连接，不仅可存取及分享 SMB 的资料夹及打印机，还可以整合入 Windows Server 的网域、扮演为网域控制站（Domain Controller）以及加入 Active Directory 成员。

Samba 是许多服务以及协议的实现，其包括 TCP/IP 上的 NetBIOS（NBT）、SMB、CIFS（SMB 的增强版本）、 DCE/RPC 或者更具体来说 MSRPC（网络邻居协议套件）、一种 WINS 服务器（又称 NetBIOS Name Server（NBNS））、NT 域协议套件（包括 NT Domain Logons、Secure Accounts Manager（SAM）数据库、Local Security Authority（LSA）服务、NT-style 打印服务（SPOOLSS）、NTLM 以及近来出现的包括一种改进的 Kerberos 协议与改进的轻型目录访问协议（LDAP）在内的 Active Directory Logon 服务）。

6. MySQL

MySQL 是一个开放源码的小型关系数据库管理系统，开发者是瑞典 MySQL AB 公司用 C 和 C++编写，并使用了多种编译器进行测试，保证源代码的可移植性。在 2008 年 1 月 16 日被 Sun 公司收购，而后 Sun 又被 Oracle 收购。目前 MySQL 被广泛地应用在 Internet 上的中小型网站中。由于它具备体积小、速度快、总体拥有成本低，尤其是开放源码这一特点，许多中小型网站为了降低网站总体拥有成本而选择了 MySQL 作为网站数据库。其具有以下特点[①]：

（1）支持 AIX、BSDi、FreeBSD、HP-UX、Linux、Mac OS、Novell Netware、NetBSD、OpenBSD、OS/2 Wrap、Solaris、SunOS、Windows 等多种操作系统。

（2）为多种编程语言提供了 API。这些编程语言包括 C、C++、C#、Delphi、Eiffel、Java、

① http://dev.mysql.com/doc/refman/5.1/zh/index.html

Perl、PHP、Python、Ruby 和 Tcl 等。

（3）支持多线程，充分利用 CPU 资源，支持多用户。

（4）优化的 SQL 查询算法，有效地提高查询速度。

（5）既能够作为一个单独的应用程序应用在客户端服务器网络环境中，也能够作为一个库而嵌入到其他的软件中。

（6）提供多语言支持，常见的编码如中文的 GB 2312、BIG5，日文的 Shift_JIS 等都可以用作数据表名和数据列名。

（7）提供 TCP/IP、ODBC 和 JDBC 等多种数据库连接途径。

（8）提供用于管理、检查、优化数据库操作的管理工具。

（9）可以处理拥有上千万条记录的大型数据库。

7．BIND（DNS 服务器软件）

Bind（Berkeley Internet Name Domain）是一款开放源码的 DNS 服务器软件，由美国加州大学 Berkeley 分校开发和维护的，它是目前世界上使用最为广泛的 DNS 服务器软件，支持各种 UNIX 平台和 Windows 平台。

8．Sendmail（电子邮件服务器）

Linux/UNIX 下的老牌免费邮件服务器，已被广泛的应用于各种服务器中，它在稳定性、可移植性、及确保没有 bug 等方面具有一定的特色，且可以在网络中搜索到大量的使用资料。

9．OpenSSH 和 OpenSSL

OpenSSH（Open Secure Shell）是使用 SSH 透过计算机网络加密通信的实现，它是取代由 SSH Communications Security 所提供的商用版本的开放源代码方案，目前 OpenSSH 是 OpenBSD 的子计划。

OpenSSL 包含一个命令行工具用来完成 OpenSSL 库中的所有功能，它是一个强大的安全套接字层密码库，Apache 使用它加密 HTTPS，OpenSSH 使用它加密 SSH，还是一个多用途的、跨平台的密码工具。

10．Apache（网页服务器）

Apache 是开源软件世界使用排名第一的 Web 服务器由软件，它可以运行在大多数计算机平台上。Apache 源于 NCSAhttpd 服务器，经过多次修改，现已成为世界上最流行的 Web 服务器软件之一。而且不断在开源社区为它开发新的功能、新的特性、修改原来的缺陷。同时，Apache 的特点是简单、速度快、性能稳定，并可做代理服务器来使用。据 Netcraft2010 年 8 月 11 日的数据统计表明，全球目前已有 119 664 128 个网站使用 Apache；Apache 市场占有率为 54.90%，IIS 为 25.87%。

小　　结

本章几乎浓缩了与开源软件相关的规定、知识、发展状况和应用情况，也全面总结了开源软件目前所处的状态。但开源软件也面临质量、完整性和安全性上的问题挑战。对此，由 U.S. Department of Homeland Security 资助的开放源码改进计划 Coverity Scan，并且 Coverity

Scan 在 2009 年报告的开放源码的质量报告显示：开放源码软件的总体完整性、质量，与安全性都有提升，近三年来的瑕疵比例降低了 16%。此外，开放源码的开发者也相当积极的在改进软件质量（2006 年以来，Coverity Scan 总计排除了 11200 多个有瑕疵的开放源码程序）。而由 Coverity 所认证的开放源码计划的完整度（Integrity Rungs）也年年在进步，例如，2009 年得到 Rung 1 认证的计划较 2008 年增加了 32%，Rung 2 则增加一倍。OpenPAM、Ruby、Samba 等计划则是已经开始在做 Rung 3 的认证。Rungs 的数字越高，也代表软件的完整度越好。

本章根据在概述出开源软件定义之前，深入分析总结自由软件的定义，并且比较自由软件与开源软件之间的异同。直接从开源软件的发展和应用状况进行了概述，并采用了相关数据进行了佐证，罗列了开源软件在发展状况、进程和商业发展模式，同时，也指出国内开源软件发展所面临的问题。直阐对开源软件分类进行了总结，并概述了开源软件目前的优点。最后从开源软件的成本、成熟度测评、选择策略、管理机制和知识产权，以及目前最有价值的十款开源软件等角度概述了开源软件的特点。

参 考 文 献

[1] 蔡俊杰. 开源软件之道. 北京：电子工业出版社，2010.

[2] 万江平，李德杰. 2020 自由开源软件发展蓝图综述. 计算机应用研究，2009，26(11)：4027-4030.

[3] 陈雪飞. 用户使用开源软件态度的区域差异调查. 图书情报工作动态. 2010(5)：13.

[4] 陆首群. 参与开源社区并不难. 信息系统工程，2007，163(7)：35-37.

[5] 李国和. 2000—2009 年我国开源软件文献计量分析. 图书情报工作，2010，54(2)：50-54.

[6] Lawton M, Notarfonzo R. Worldwide Open Source Software Business Models 2007–2011 Forecast：A Preliminary View. IDC Inc.

[7] 工业和信息化部软件与集成电路促进中心知识产权部. 开源软件涉及的相关知识产权问题分析. 中国集成电路，2010，137(10)：77-89.

[8] 张平，马骁. 开源软件对知识产权制度的批判与兼容（二）——开源软件许可证的比较研究. 科技与法律，2004，2：46-58.

[9] 姬学. 开源软件商标许可政策. 网络法律评论，2010，10(1)：119-126.

[10] Diomidis Spinellis, Georgios Gousios, Vassilios Karakoidas, etc. Evaluating the Quality of Open Source Software. Electronic Notes in Theoretical Computer Science，2009，233(27)：5-28.

第 2 章　面向开源软件的软件架构原理

软件构架目标是在软件生命周期里使软件具备可靠性（Reliability）、安全性（Security）、可伸缩性（Scalability）、可定制化（Customizability）、可扩展性（Expandability）、可维护性（Maintainability）、客户体验/可用性（Customer Experience/Availability）、市场时机/市场需求（Time to Market/ Market Demand）。同时，根据 IEEE 标准指出，软件架构的活动则代表了定义、记录、维持、改进一个软件构架并确保其正确执行的一系列活动。而面向 Web 开源软件的软件开发技术是一门新型的软件构架模式和开发技术，也需要审视其软件构架原理、具体的实施方法、软件构架活动和所需要满足的软件构架目标等，并能选择满足各类业务需求和不同类型用户要求的开源软件作为面向开源软件构架的研发技术。因此，面向开源软件的软件构架除了能满足软件构架基本要求外[1]，还需要满足近年来对软件架构的协同和可信要求，从而形成一种面向开源软件的软件架构体系结构。

2.1　软件构架概述及基本方法

一般认为，软件架构（Software Architecture，SA）是一系列有关软件整体结构与组件的抽象描述和建模，用于指导大型软件系统各个方面的设计，即软件架构是一个系统的草图。软件架构描述的对象是直接构成系统的抽象组件，且让各个组件之间的连接明确和相对细致地描述组件之间的通信。在实现阶段，这些抽象组件被细化为实际的业务组件，比如面向对象分析设计中就是具体某个类或者对象。而在面向对象组件（领域）中，组件之间的连接通常用接口来实现，使其成为一个业务逻辑实体。同时，软件体系结构是构建计算机软件实践的基础，与建筑师设定建筑项目的设计原则和目标，以及绘制的建筑草图是一样的。从而使得在软件的构架中，需要由软件构架师来实施，而一个软件构架师需要有着广泛的软件理论知识和相应的经验来实施和管理软件的体系构架、定义和设计软件的模块化，模块之间的交互，用户界面风格，对外接口方法，创新的设计特性，以及高层事物的对象操作、逻辑和流程。最终使软件架构师或者系统架构师所描述的软件构架作为满足不同客户需求的实际系统设计方案。这时，根据 IEEE 1471 定义：

（1）架构是描述组件间和组件与环境间的关系，并在引导与设计原则中体现系统的基本结构。

（2）系统是为实现某个（些）特殊作用的组件的集合，而一个系统是为了实现一个或多个任务而存在。

（3）环境决定了开发，操作，策略和其他影响系统的设置和条件。

（4）任务是指系统为了实现对对象设置的使用或操作。

（5）涉众是对于系统有利益关系或关注的个人，团队或组织，而软件架构是为了实现涉众的需要而创造的，但一般情况下不可能满足所有的需求。

计算机系统的软件架构是包含软件部分，外部可见特性部分和它们之间关系的系统结构；架构是系统的组织结构和相关行为，且架构可被重复分解为接口（组件定义了所需接口的行为），互联部分的关系和结合部相互作用的部分；而通过接口相互作用的部分包括类，组件和子系统。软件架构或系统由组成系统的结构的相互作用和软件结构的重要设计组成；其中，设计决定成功实现需求所期望满足的质量，以及决定为系统开发、支持和维护提供概念上的基础。总之，软件构架定义了软件的结构和行为，以及与之相关联的作用和联系；架构风格定义了组件和连接型的语法，和连接的方法。

同时，软件构架也不是孤立于软件建造的本身，还会受到来自各方的支持和影响。由软件定义可以明白：架构定义了一组连贯的相关元素，这些元素来自不同的领域、不同的需求、不同的支撑、不同的团队等；这足以说明软件构架是一个复杂、集成化的系统。又根据 IEEE 12207 指出，一个系统是包含了一个或多个进程，硬件，软件，工具与可以满足需求的集合。因此，从构架范围类型可以分为：

（1）硬件架构——包括 CPU、内存、硬盘、周边设备（例如打印机等），以及与之连接这些元素的部分。

（2）组织架构——是由一些关于商业进程，组织结构，规则和职责，以及与组织核心能力的部分构成。

（3）信息架构——是指包含组织好的信息结构。

（4）技术构架——是指所选择的技术体系，即在软件构架过程中所需求的技术体系。

（5）团队构架——是指在实现一个软件系统时，所需的人员构成，包括管理人才、技术人才等。

但软件架构，硬件架构，组织架构、技术构架、团队构架和信息架构是全部系统架构的子结构，图 2-1 展现了软件构架各个方面[4]。

从图 2-1 中的框结构可以发现软件构架的特征：

（1）一个系统有一个构架，一个构架需要综合多方面的资源和知识。

（2）一个系统完成一项任务，这里所谓任务的意思是一个任务由多个功能组成。

（3）一个系统存于一个环境中，并受这个环境的影响，也受该环境约束；并且环境的合理性、和谐性和有效性直接能使系统运行于所在的环境中，为其创造价值。

（4）一个系统有一个或多个涉众，所谓的涉众包括在一个软件系统生命周期内所有参与这个系统的人员及相关的人员。

（5）一个构架对应一条构架描述，这里主要指架构描述语言（Architecture Description Language，ADL；详见 2.2.1.3 节第（2）点），它是“软件工程”和“企业建模、工程”两个团体的语言术语。在“软件工程团体”中，它是一种计算机语言，用来描述软件或系统架构；若是架构是技术性架构，则该架构必须被清楚的传达给软件开发者。在“企业建模、工程团队”中，也有基于企业建模和工程的语言（如 ArchiMate①、DEMO 等）；若是架构是功能性架构，则该软件架构必须被清楚的传达给利益相关者和企业工程师。

（6）一条构架描述、识别一个或多个涉众。

（7）一条构架描述、识别一条或多条关联。

---

① http://www.opengroup.org/archimate/doc/ts_archimate/

（8）一条构架能描述所提供的理由。

（9）一个涉众有一条或多条关联，并且一条关联对一个或多个涉众都很重要。

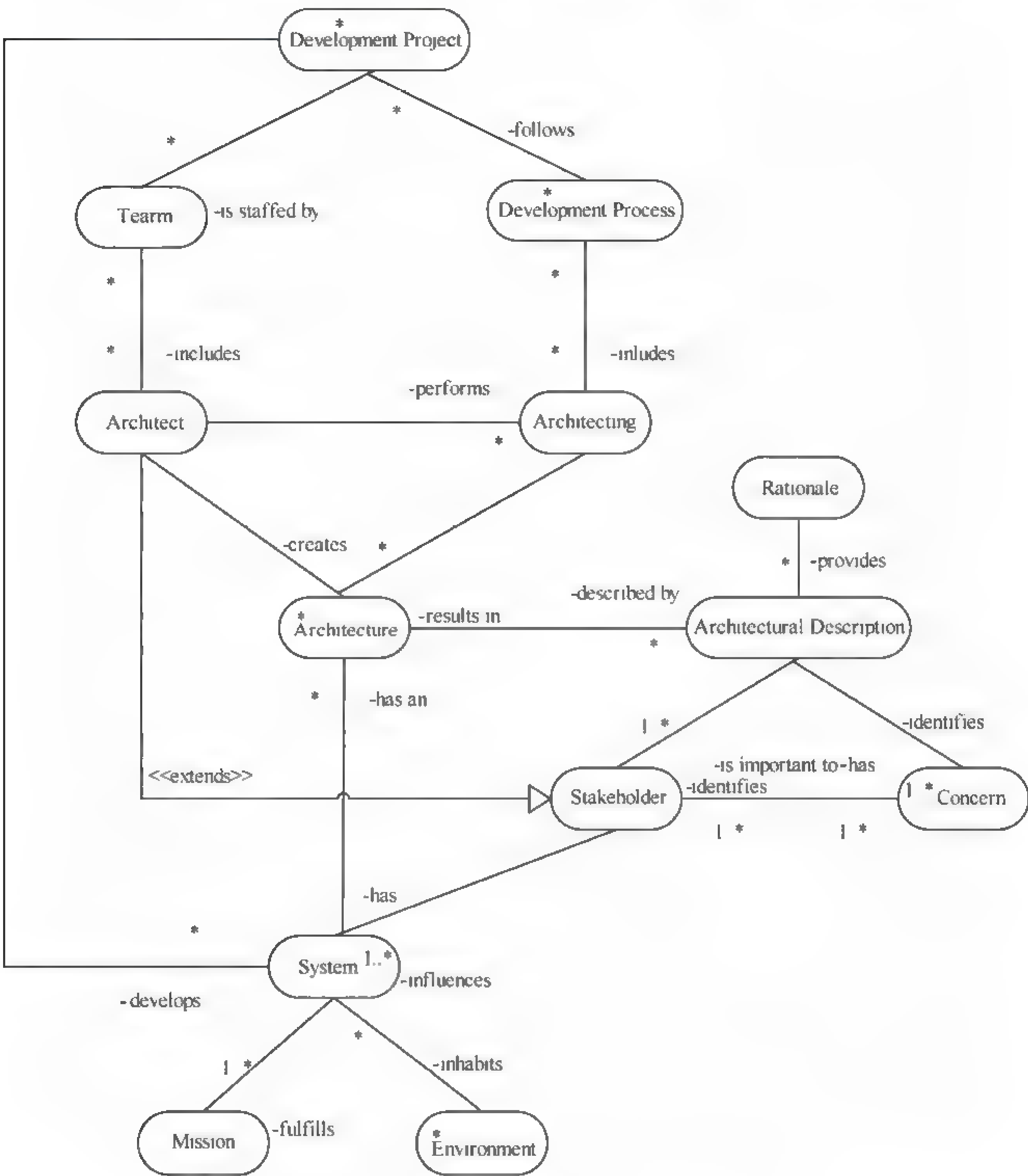

图 2-1　软件架构相关术语的模型（来源：IEEE 1471）

在图 2-1 中非框图可以发现软件构架另外特征：

（1）一个团队组负责一个开发项目。

（2）一个开发项目遵循一个开发流程。

（3）一个开发项目交付给一个系统。

同时，项目组里包含一个架构师（架构师即是技术主管，即构架师不但具备丰富的知识，还应具备领导才能，而且架构师的领导能力在团队中和项目质量控制中起着十分重要的作用）和一个项目经理（项目经理是来管理项目的资源，时间进度和花费的）。构架师的主要职责和

特点是：负责软件架构，即也是涉众中的一种，最终架构得出一个软件构架和创造出一个软件构架。

下面就从软件构架基本特点、构架质量分析、软件架构“4+1”视图模型、软件构架师和软件构架在面向开源软件的构架中的作用等五个角度对软件构架进行分析和概述。

2.1.1　软件构架的特点

软件构架是整个软件系统生命周期内的基础，它的合理程度直接影响整个软件系统的可靠性、安全性、可伸缩性、可定制化、可扩展性、可维护性、可用性和可需求性。因此主要具有以下一些特点：

（1）软件构架不但是一门软件科学，还是一门艺术，这是因为软件构架需要有创造力和创新思维。

（2）软件构架包含软件结构和行为，同时，在结构和行为的基础更需要关注所构成和实现的元素（即整个软件系统的组成部分）。

（3）软件架构是为了实现涉众的需要而创造的，因此，架构可以平衡涉众需求，以减少各主需求互斥和冲突。

（4）软件架构体现软件系统的决策，并能做出符合一个架构的样式，从而体现一种软件架构风格（架构风格定义了组件和连接型的语法，和连接的方法[5]）。

（5）软件构架受所在的环境和所在的团队影响，这些因素会直接导致整个软件构架的成熟度。反过来环境和团队也受软件构架影响和制约。

（6）一个软件构架直接呈现在软件系统中，且拥有一个特定的应用范围。

（7）软件架构跨越很多学科，即一个软件架构涉及多种学科，需要多学科的知识加以融合，最终才能有效实现软件架构。

（8）架构是时刻进行的行为，当一个软件的原型完成时，允许根据需求变化而进行修正和适当的更改。变化曲线如图 2-2 所示：

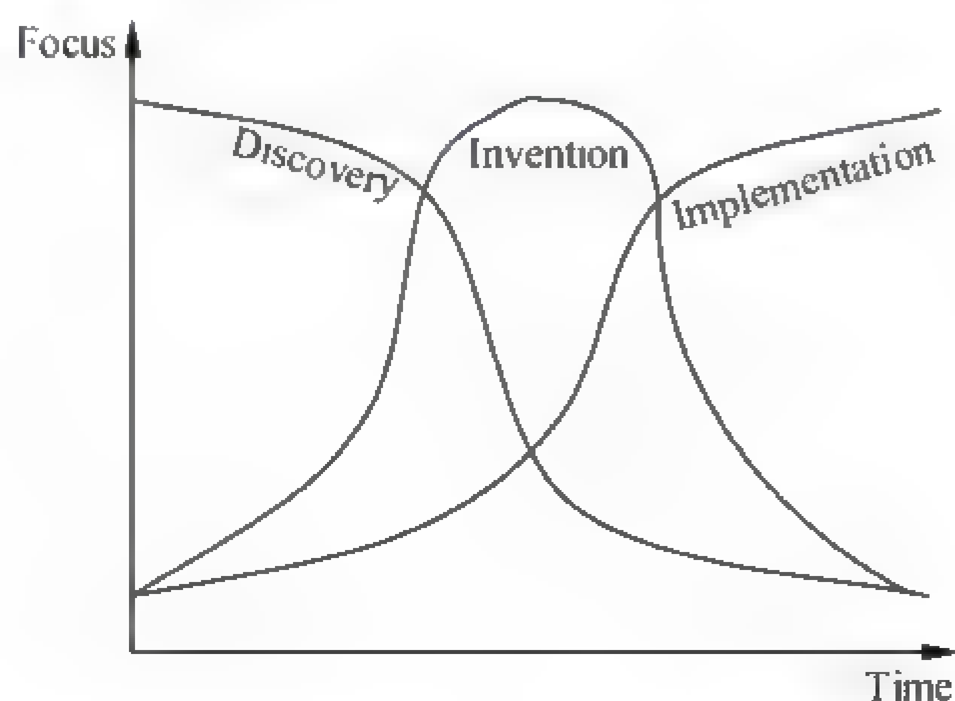

图 2-2　软件构架的行为随时间变化（由 IBM 杰出工程师 Bran Selic 绘制）

（9）由于一个软件构架满足许多涉众的需求，因此，软件架构受涉众所驱动、影响和制约。

（10）软件构架需要在需求和影响因素中寻求平衡和折衷，并根据自己过去的有益经验来降低软件构架的风险。

（11）软件架构既是自上而下也是自下而上的。

这些软件构架的特点决定一个软件系统的构架是一个系统性的工程，受制多方面的因素，需要从系统的角度去创造和创新，才有可能构造一个满足各方的软件体系结构。

2.1.2 软件构架的质量评估

软件构架的质量评估主要由管理构架、支撑构架、业务构架、技术构架和维护构架等五部分组成，以及还包括后期的代码质量分析，并以此建立软件构架质量评价模型来指导构架质量，而对质量的定义是多样的，一般认为，质量对于实现成功结果具有非常有效的重要性，质量越高，项目成功的机会就越大。质量的实现要确定质量目标，同时也要确定增加功能和缩短日程目标，使这三者结合应用来确立最终的质量目标，并且使用有效的方法来减轻了这三个方面之间的依赖性，使它们之间相互独立，又能相互联系。而质量属性主要刻画特定上下文质量的元素，例如性能、安全性、可移植性、功能等，其中每个属性都不是绝对量，它们的相关性直接与给定的情形联系在一起。为了能够正确测量属性，必须进行进一步的任务分解，进一步细分到每一个场景，从而使需求和质量属性之间的关系有助于了解软件体系结构的适用性，最终用质量属性来限定特定的场景，并进一步使这些场景可以用作软件设计细化的理想工具。

而评估的最终的目的是最有效的满足用户的需求。毕竟一个软件设计得再出色，使用的技术再先进，只要不能满足用户的要求，也是没有什么意义可言的。因此，在实行软件构架质量评估时，需要以用户需求为中心来设计。一般的，高质量构架是高质量软件的前提，而要评估质量的优劣，需要满足以下几个方面要求：

（1）最终的软件产品要满足用户的需求，且具备很好的可用性和易用性。

（2）通过需求有效把握软件构架的合理进度、成本、功能和维护的关系，并通过计划、管理、控制、维护使它们之间的关系存在一定的平衡，以保证整个软件生命周期产出和收益最大化。

（3）由于一个需求往往是随时间变化的（如图 2-2），怎么保证一个软件构架保持一定的稳定性，这就需要软件构架具备扩展性和灵活性，并能够适应一定程度的需求变化。

（4）一般情况下，一个构架内，主动功能的工作量并不大，真正的工作量是来自构架外的各种要求，而往往一个软件系统的健壮性和稳定性是受构架外的功能变化影响。因此，软件架构应具备有效处理构架外的各种变化。

（5）软件构架需求保持成本和性能平衡，这是因为性能往往取决于各方的非功能需求，而且非功能性在任何地方都是存在的，这时需要合理处理好性能在各方面中的平衡关系。

（6）软件构架也需要满足可持续化发展，这样可以有效降低重复开发，重复做同样的事。当满足可持续发展后，就可以为后续发展作贡献，这是一个优秀构架的前提，也是为后续的高质量软件系统的保障。

（7）软件构架最终实现就是需要高质量的代码来实现，这项在软件生命周期内也是非常重要的，即构架得再好，设计得再完美，各方面平衡得再合适，没有高质量的代码来实现也是无法满足用户需求的，也无法呈现给需求者。因此，高质量的代码是软件构架和最终的实现中间过程，也是最重要环节之一。

这些因素的是影响高质量的软件构架的关键，也是影响后续的软件系统质量的关键，这是因为一种合理的质量评估方法可以有效帮助分析软件体系结构设计是否适合于一组给定的需求。而需求提供了用以确定质量预期的上下文，若软件构架及软件的需求得到满足，那么其质量目标也应该会得到满足，也能有效把握质量对整个软件生命周期的影响和约束。下面从卡内基梅隆大学的软件工程协会（Software Engineering Institute，SEI[①②]）定义的五种软件体系结构（软件构架）评估方法和软件构架的代码质量两个角度进行分析。其中 SEI 五种方法包括质量属性专题研讨会（QAW）、体系结构权衡分析方法（ATAM）、软件体系结构分析方法（SAAM）、积极的中间设计审核（ARID）和属性驱动设计（ADD）。QAW 在定义体系结构之前执行，ARID 在设计工作过程中执行，而 ATAM 和 SAAM 则在已经完成体系结构之后执行。QAW 最适合于(Rational Unified Process，RUP)的初始阶段；ARID 应该在 RUP 的细化阶段中执行；ATAM/SAAM 工作可以在该生命周期中需要体系结构审核的任何地方进行。SEI 评估方法和软件构架代码质量的方法和角色如表 2-1 所示。

表 2-1　SEI 四种质量评估方法与角色

方法	角色	过程	阶段
QAW	软件分析人员	需求分析	初期
ARID	技术审核和选择人员	分析设计	细化
ADD	软件设计师	分析设计	再细化
ATAM/SAAM	软件架构师	分析设计	构造

使用评估方法可以促进对当前体系结构（软件架构）设计更好的了解需求状况，以及需求变化，并支持更高效地确定软件体系结构中的质量。同时，将需求与场景驱动的质量属性进行匹配，可以促进更准确的软件体系结构，从而提高所构架的软件系统的质量和生命周期。

2.1.2.1　SEI 评估方法

研究质量、质量属性和软件体系结构之间的动态关系，是以更好地了解怎样能够提高软件质量和如何高效地实现所确定的目标被有效控制软件构架的必由之路，也是软件系统得以实现且满足用户需求必要的保证。

（1）质量属性专题研讨会（QAW）

QAW（Quality Attribute Workshop）[③]是一种用于在创建软件体系结构之前发现质量属性的方法；因此，它为软件构架提供初期的工作，也可以认为 QAW 处于整个软件生命周期顶端，如软件构架前期的性能、安全性、维护成本等特定质量的实现都需要高度依赖于设计良好的软件体系结构。而 QAW 引出活动是在由协调人员和系统参与者组成的专题讨论会中执行的。同时，通过聚集大家的智慧来避免造成后期灾难性的困境，以及安全所带来的威胁。但这个部分对于软件结构中的任何一功能来说都是不同的，因为不同的软件功能的业务不同，使得业务架构的目的对业务建模和抽象也是不同的，当然所考虑的可重用、可扩展的部分也是不同的。同时，也不得不考虑中后期软件设计师、程序员的工作情况，以及对整个软件构

① http://www.sei.cmu.edu

② http://www.ibm.com/developerworks/cn/architecture/ar-qualassess.html

③ http://www.sei.cmu.edu/publications/documents/03.reports/03tr016.html

架的影响。QAW 工作的输出是一个体系结构驱动因素列表、场景，以及一个经过优先排序的场景列表和细化的场景等，如图 2-3 所示。其实现步骤如表 2-2 所示。

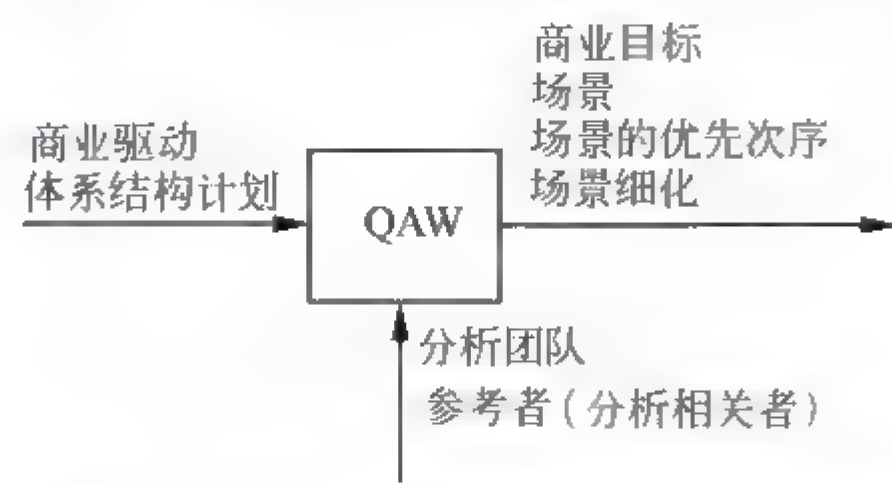

图 2-3　QAW 的输入、输出与参与者

表 2-2　QAW 实现步骤

角色：系统分析		
输入：商业案例[商业驱动]		
前景文档[体系结构计划]		
输出：商业案例[QAW 优化商业目标，并将结果反馈给业务案例]		
补充说明[作为库的场景]		
步骤	**描述**	**操作**
Step1	QAW 陈述	QAW 协调人员描述专题讨论会的理论基础、QWE（Quality Attributes Expectations）涉及的步骤和该工作中的预期情况
Step2	业务和使命陈述	某个参与者陈述系统的业务和使命驱动因素，协调人员并捕获相关业务信息
Step3	体系结构计划陈述	在解决方案的软件生命周期（Software Life Cycle，SLC）中的这一方面，可能不存在详细的系统体系结构，或可能具有大致的描述、关系图或其他附带技术细节的元素；这时需要某个技术参与者向与会人员陈述这些内容；并且协调人员继续捕获重要的方面以便以后分析
Step4	确定体系结构驱动因素	协调人员临时退出讨论并整理笔记；向参与者陈述所记录的重要体系结构驱动因素以达成共识
Step5	场景自由讨论	一旦就体系结构驱动因素达成一致，协调人员将充当场景生成活动的召集人；每个参与者定义满足其所关注方面的场景；至少执行两个回合的表决；协调人员确保每个体系结构驱动因素至少存在一个场景
Step6	场景合并	协调人员向参与者询问可能的场景合并，从而更好地集成成一个更可靠的场景
Step7	场景优先排序	由参与者驱动的所需结果是一组目标，这些目标按照对于边项目的重要性进行优先排序
Step8	场景细化	细化最重要的四个或五个场景（取决于时间），阐明这些场景的刺激因素、响应、刺激源、环境、所刺激的构件和响应度量
工作流详情：		
要求		
明白与之相关用户的需求		

（2）属性驱动设计（ADD）

ADD（Attribute Drive Design）也是一种定义软件架构的方法，该方法将分解过程建立在软件必须满足的质量属性之上，如图 2-4 所示。

ADD 位于需求分析之后，操作步骤如表 2-3 所示。

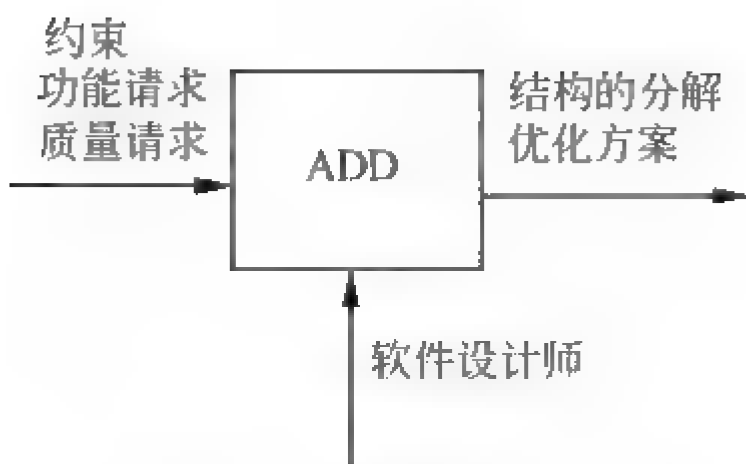

图 2-4 ADD 的输入、输出与参与者

表 2-3 ADD 操作步骤

角色：软件设计师[软件设计师]
输入：前景[约束]
验证结构概念[约束]
用例模型[功能要求、质量要求]
补充说明[质量要求]
结果：软件结构文档[体系结构的模块分解表示，获得并发性和部署的观点]

步骤	描述	操作
Step1	模型分析	选择所要分析的模型，为属性驱动分析设计创造条件
Step2	完善模块	a. 选择结构驱动。 b. 选择满足体系结构驱动的结构模式。 c. 从用户案例角度初始化模块和分配功能，其目的是用多视图代表结果。 d. 定义子模块的接口。 e. 验证和提炼用户案例和质量场景，并且使其为子模块约束
Step3	重复 Step1、2	重复 Step1、Step2 直到满足条件，再转向下一模块

流程详情：
分析设计
定义一个候选结构
执行结构合成

（3）体系结构权衡分析方法（ATAM）

通过 QAW 方法将获得前期的体系结构文档和质量元素进行优先排序后，交付给 ATAM（Architecture Tradeoff Analysis Method）[6]。ATAM 不仅描述某个体系结构对特定质量目标的满足程度，而且也用于执行体系结构审核以评估当前体系结构对其业务驱动因素的适用性。而且还提供了对哪些属性在体系结构质量中所具有的权衡认识。同时，在 ATAM 中，可以重用 QAW 的体系结构文档来交付清晰的质量属性定义，如图 2-5 所示。

其操作步骤如表 2-4 所示。

（4）软件体系结构分析方法（SAAM）

SAAM（Software Architecture Analysis Method）是用于分析软件体系结构且具有文档记录的最早方法之一，它提供了一种将可测量的质量属性场景附加到一般属性声明的方法，并且提供了比 ATAM 更简单的方法，这就使得有些 SAAM 步骤与有些 ATAM 步骤相匹配。其操作步骤如表 2-5 所示。

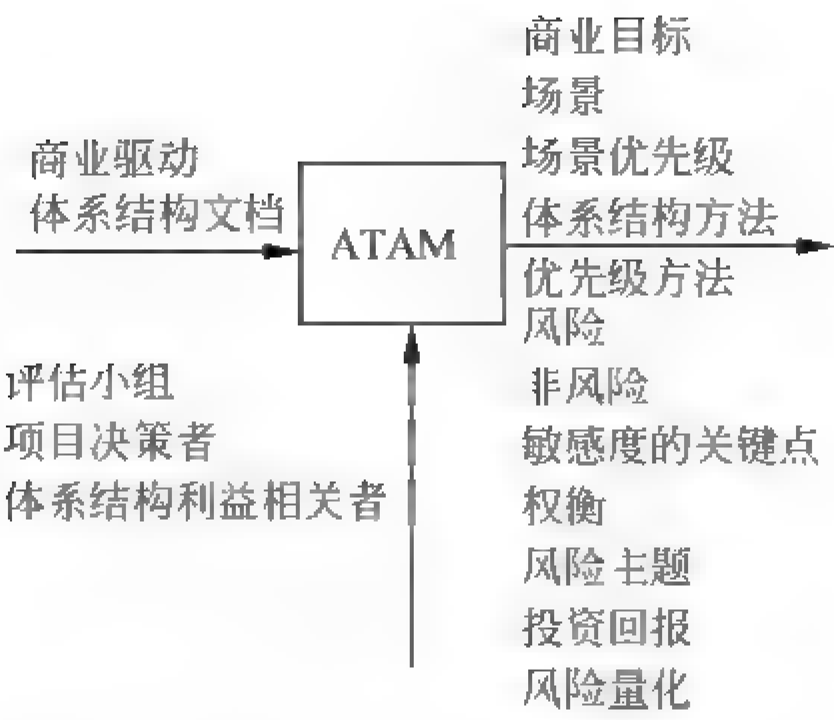

图 2-5 ATAM 的输入、输出与参与者

表 2-4 ATAM 操作步骤

角色：技术评审[评估小组]
输入：商业案例、前景[商业驱动]
　　　软件结构文档[结构文档]
输出：评审记录[补充在进行的商业目标上的风险主题和影响]
　　　软件结构文档[附加说明关键点和权衡]
　　　补充说明[场景]

步骤	描述	操作
Step1	陈述 ATAM	陈述 ATAM 理论基础和工作要求
Step2	陈述业务驱动因素	参与者陈述系统的业务和使命驱动因素，并协调人员捕获相关业务信息
Step3	陈述体系结构	项目架构师陈述体系结构，并陈述该体系结构是如何满足业务驱动因素的
Step4	确定体系结构方法	陈述所要处理的驱动因素；架构师确定软件体系结构设计所采用的具体方法，并且协调人员对这些方法做文档记录，以便为构架师、软件设计师提高设计参考和功能来源
Step5	生成质量属性功能树	更好地可视化和组织与项目相关的质量属性； QAW 步骤在这里会非常有用，为生成质量属性功能树提供具体的文档和质量元素； 一直对属性进行分解，直到分解为支持这些属性的场景
Step6	分析体系结构方法	将 Step 4 中发现的体系结构方法与处于质量属性功能树树叶上的场景作比较，以更好地了解所采取的方法是否与参与者的驱动场景相匹配，从而最终确定权衡点和评估风险
Step7	自由讨论并优先安排场景	为了确保没有被忽略的重要细节，与参与者一起进行另一回合的场景发现活动。对发现结果进行优先排序（举行一轮表决）并做文档记录
Step8	分析体系结构方法	为获取可能存在新的场景，因此再次执行 Step 6 的活动，并选择具有高优先级的场景
Step9	陈述结果	向参与者陈述信息——方法、场景、权衡、风险，其目的是做出有关该体系结构和参与者需求的适用性决策

工作详情：
　　分析设计
　　　　完善结构

表 2-5　SAAM 操作步骤

步骤	描述	操作
Step1	确定场景	在所确定的场景中开发场景
Step2	明确体系结构	在明确体系结构的前提下，描述体系结构
Step3	分类与排序	在所确定的场景和体系结构中，对场景进行分类和优先排序
Step4	评估场景	分别评估间接场景，即某个场景与 Step 2 中描述的体系结构方法之间不存在匹配
Step5	场景交互	协调人员向参与者询问可能的场景合并，并进行各个场景有效的交互，从而更好地陈述更可靠的场景

（5）活动的中间设计审核（ARID）

ATAM 和 SAAM 都集中于完成后的体系结构，ARID（Active Reviews for Intermediate Designs）[①]方法集中于未完成的体系结构，它是一种新的体系结构方法：它的优点在于等待体系结构设计完成即可了解该设计是否在沿正确的方向进行，如图 2-6 所示。其操作步骤如表 2-6 所示。

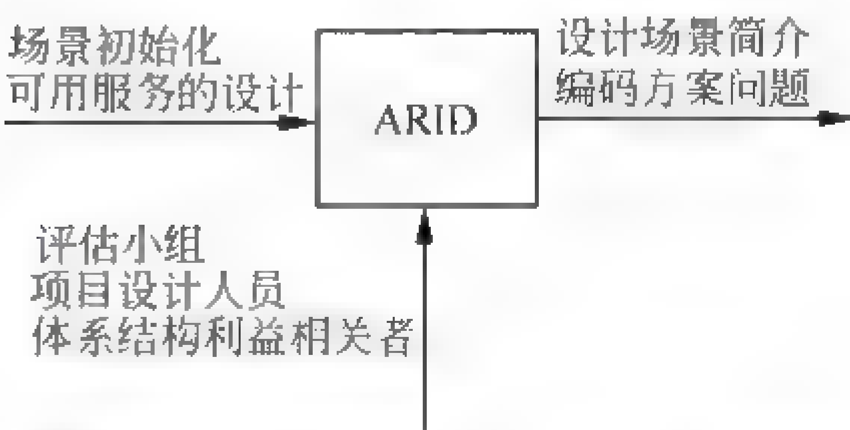

图 2-6　ARID 的输入、输出与参与者

表 2-6　ARID 操作步骤

角色：技术评审[评审小组、体系结构利益相关者]

输入：软件体系结构文档[服务可用性的设计]

　　　补充说明[场景初始化]

结果：评审记录[附加说明问题]

步骤	描述	操作
Step1	确定审核人员	ARID 中的审核人员是设计参与者，以及参与了可靠设计投资的各方人员
Step2	准备设计讲座	其目标是设计人员详细地陈述所需求的设计，以使审核人员能够使用该设计
Step3	准备初始场景	由协调人员和设计人员执行
Step4	准备资料	准备所需要的审核资料
Step5	陈述 ARID	陈述 ARID 功能和工作要求
Step6	陈述设计	陈述 ARID 的设计及相关流程
Step7	对场景进行自由讨论和优先排序	对场景进行自由讨论和优先排序，这是由于场景用于涵盖适当的需求范围
Step8	应用场景	从优先排序列表中的最高位置的场景开始应用，审核人员使用伪代码来验证场景的适用性
Step9	总结	对评估工作的结果做文档记录并交付给适当的参与者

工作流程详情：

　分析设计

　　完善体系结构

　　分析行为

① http://www.sei.cmu.edu/publications/documents/04.reports/04tr011.html，http://www.sei.cmu.edu/reports/00tn009.pdf

2.1.2.2 软件架构的代码质量

代码质量情况主要由代码的耦合度来度量，其通常用整数来表示几个相关对象的度量，然后用该相关对象度量值来表示代码的传入耦合和传出耦合度。其特点是：高传入耦合度表示对象具有太多的职责，而高传出耦合度表示对象不够独立。

传入耦合度（C_a）可由测量抽象性实行进一步检查，即通过理解传入耦合来表示软件构架代码组件的职责，并持续监视这个职责，以防止软件架构出现熵（Entropy），这里的熵是指传入耦合体系的混乱的程度。然而，熵容易出现在大多数设计良好的系统中，但可以通过对熵的控制来进一步提高软件构架代码设计的灵活度。

传出耦合（C_e）通常依赖于某个特定组件的组件，即传出耦合则是某个特定组件所依赖的一些组件，但可以把传出耦合看作传入耦合的逆转。

这时，将系统的传出耦合和传入耦合的数量结合起来，形成另一个度量，叫做测量不稳定性。即通过将传出耦合除以传出耦合与传入耦合的和生成一个比率，如下式所示：

$$\text{rate}(C_e, C_a) = \frac{\sum C_e}{\sum (C_a + C_e)}$$

式中 $\text{rate}(C_e, C_a)$表示一个稳定的包（值接近于 0）或者不稳定的包（值接近于 1）。而在设计和实现软件架构时，依赖于稳定的包是有益的，这是由于这些包不太可能更改。

由此可以得到软件构架的代码质量是可以由耦合度来度量的，最终使其耦合度的数量可以有效度量某一软件构架的代码的质量。

2.1.3 软件架构“4+1”视图模型

基于“4+1”视图模型的软件构架模型是由 Philippe Kruchten 在 1995 年在《IEEE Software》中提出[8]①，引起了业界的广泛关注，并最终被 RUP（Rational Unified Process）采纳，现已经成为架构设计的结构标准。这个“4+1”包括逻辑视图（Logical View）、过程视图（Process View）、物理视图（Physical View）、开发视图（Development View）和场景（scenarios）或用例（use cases），如图 2-7 所示。

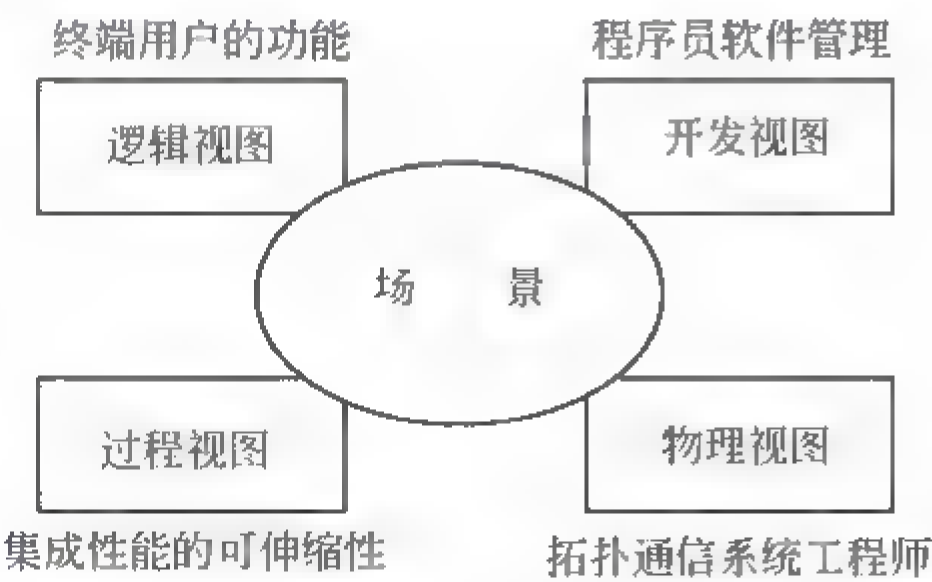

图 2-7 “4+1”视图模型图（由 Philippe Kruchten 绘制）

“4+1”视图一览如表 2-7 所示。

① http://www.ibm.com/developerworks/cn/rational/r-4p1-view/

逻辑视图：使用面向对象设计的对象模型。

过程视图：捕捉所设计的并发和同步特征。

物理视图：描述了软件到硬件的映射，反映了分布式特性。

开发视图：描述了在开发环境中软件的静态组织结构。

场景：用以说明这些视图的组织结构，就是第五视图，即“4+1”中的“1”。

表 2-7　“4+1”视图一览表[8]

视图	逻辑视图	过程视图	开发视图	物理视图	场景
组件	类	任务	模块、子系统	结点	步骤、脚本
连接工具	关联、继承、约束	会面、消息、广播、RPC 等	编译依赖性、with 语句、include 等	通信媒体、LAN、WAN、总线等	
容器	类的种类	过程	子系统（库）	物理子系统	Web
涉众	最终用户	系统设计人员、集成人员	开发人员、项目经理	系统设计人员	最终用户、开发人员
关注点	功能	性能、可用性、容错、整体性	组织、可重用性、可移植性、产品线	可伸缩性、性能、可用性	可理解性
工具支持	Rose	UNAS/SALE、DADS	Apex、SoDA	UNAS、Openview、DADS	Rose

正如前面所述，软件架构是用来处理软件高层次结构的设计和实施，而基于“4+1”视图的软件构架模型则可以有效表达这些设计的功能性与非功能的描述，以及具体实现的流程。最终以精心选择的形式将若干结构元素进行装配，从而满足系统主要功能和性能需求，并满足系统的可靠性、可伸缩性、可移植性和可用性等非功能性需求。Perry 和 Wolfe 使用一个精确的公式（简称 Perry & Wolfe 公式）来表达软件构架：

软件构架={元素，形式，关系/约束}

[Software architecture = {Elements, Forms, Rationale/Constraints}]

由于软件架构涉及到抽象、分解、组合、风格等，这时就可以采用多视图的组成模型来描述和表达它，并独立运用 Perry& Wolfe 公式在每一个视图中，用来捕捉工作形式、模式、关系和约束，从而使构架与需求连接起来。而构架的描述可以由以上四个视图来描述，并用场景进行说明或表达。这时定义构架与视图集合：

构架与视图={组件，容器，连接符}

[Architecture and View={Component, container , connector }]

2.1.3.1　逻辑视图之面向对象分析

逻辑架构主要支持功能性需求，也为各方需求提供功能方面的需求。由于一个具体系统需要分解为一系列关键抽象，其表现为对象或类，而且通过抽象、封装和继承的原理实现表达。但逻辑分解并不是为了功能分析，而是用来识别遍布系统各个部分的通用机制和设计元素。通常可以使用 Rational/Booch （见图 2-8）方法中的类图和类模板来表示逻辑架构[9]，其中，类图用来显示一个类的集合和它们的关联、使用、组合、继承等逻辑关系，并且相似的类可以划分成类集合；而类模板关注于单个类，并强调主要的类操作，并且识别关键的对象特征。这时，若需要定义对象的内部行为，则使用状态转换图或状态图来完成。

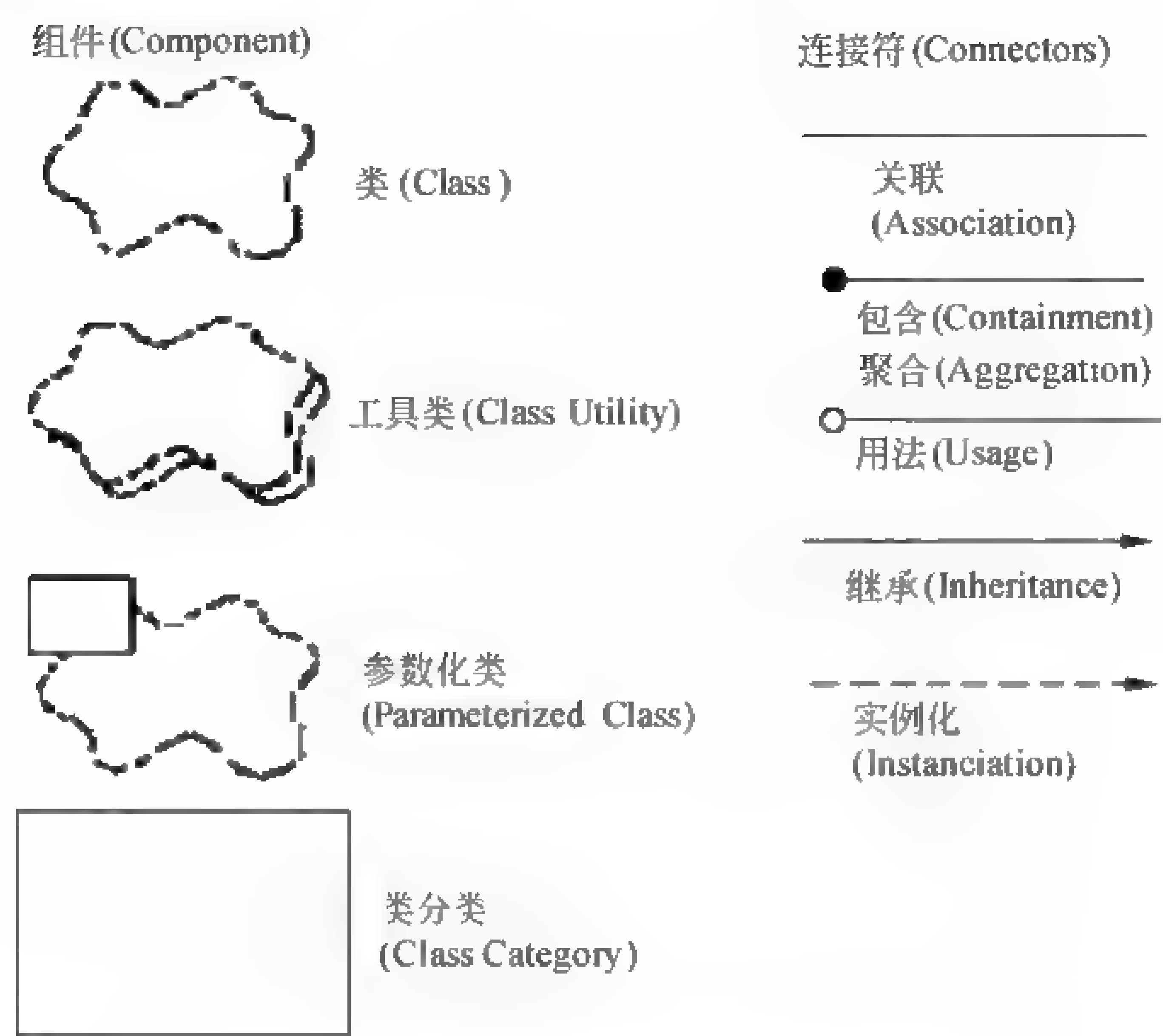

图 2-8 面向 Booch 的逻辑视图的标记法

2.1.3.2 过程视图之分解

软件构架的过程视图主要考虑性能、可用性、并发性、分布性、系统完整性、容错性等非功性需求问题，以及逻辑视图的主要抽象如何与进程结构相配合在一起，即在哪个控制线程上，对象的操作被实际执行，并且各种视图并不是完全是正交的或独立的，视图的元素根据某种设计规则和启发式方法与其他视图中的元素相关联。在具体实现过程构架时，一般分层次在抽象上进行描述，且每个层次针对不同的问题，如在最高的层次上，进程架构可以视为一组独立执行的通信程序，它们分布在整个一组硬件资源上，这些资源通过 LAN 或者 WAN 连接起来。使得多个逻辑网络可能同时并存，共享相同的物理资源。面向 Booch 的过程视图标记如图 2-9 所示。

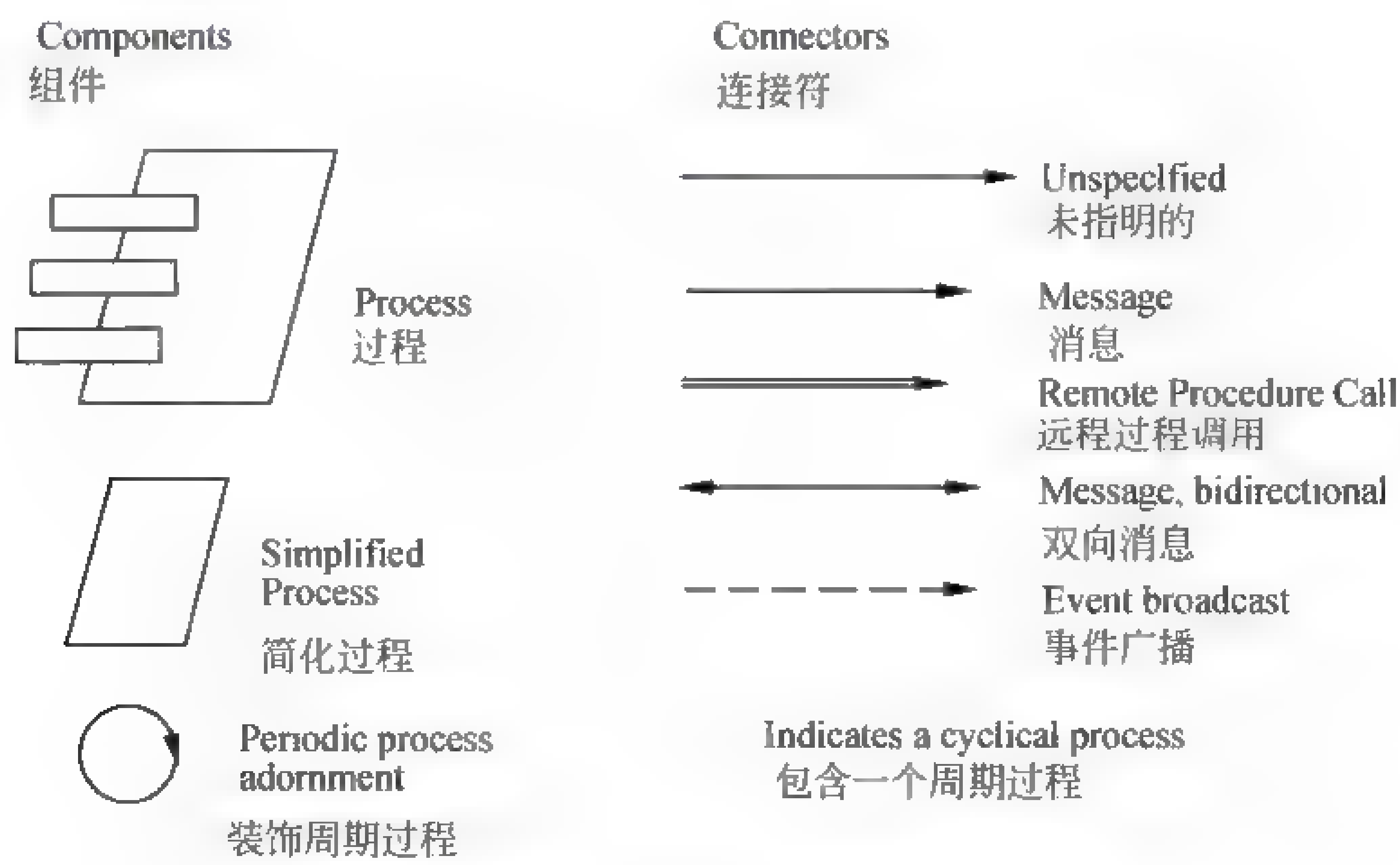

图 2-9 面向 Booch 的过程视图标记法

过程视图中的流程是由可执行的任务单元组构成的，且代表了可以进行策略控制过程架构的开始、恢复、重新配置和关闭层次流程，也可以处理分布式负载过重或重复应用过程。在进行流程处理时，是将软件划分为一系列的单独任务，由每一任务的线程在处理结点上进行调度处理。但这时需要区别主任务和次要任务，其中主要任务是可以唯一处理的架构元素，通过基于消息的同步或异步来实现各个分任务通信；次要任务是由于实施原因而引入的局部的周期性活动、缓冲、暂停等任务，一般通过会见或共享资源的方式进行通信。

2.1.3.3　开发视图之子系统分解

开发视图关注软件开发环境下实际模块的组织，它也是各种活动的开发基础，因此，开发视图一般由模块和子系统图来表达。而完整的开发视图只有当所有软件元素被识别后才能加以描述。但是，可以列出控制开发视图的分块、分组和可见性规则。在具体分解开发视图时，可以把软件打包成小的程序块和子系统，这样可以将这些小块程序分配到不同的开发人员，把子系统可以组织成分层结构，并在每个层为上一层提供良好定义的接口，最终通过这些接口实现各子系统连接及在各子系统中实现各程序块耦合。通常情况，开发视图考虑的内部需求与开发难度、软件管理、重用性和通用性及由工具集、编程语言等限制。图 2-10 是面向 Booch 的开发视图标记法。

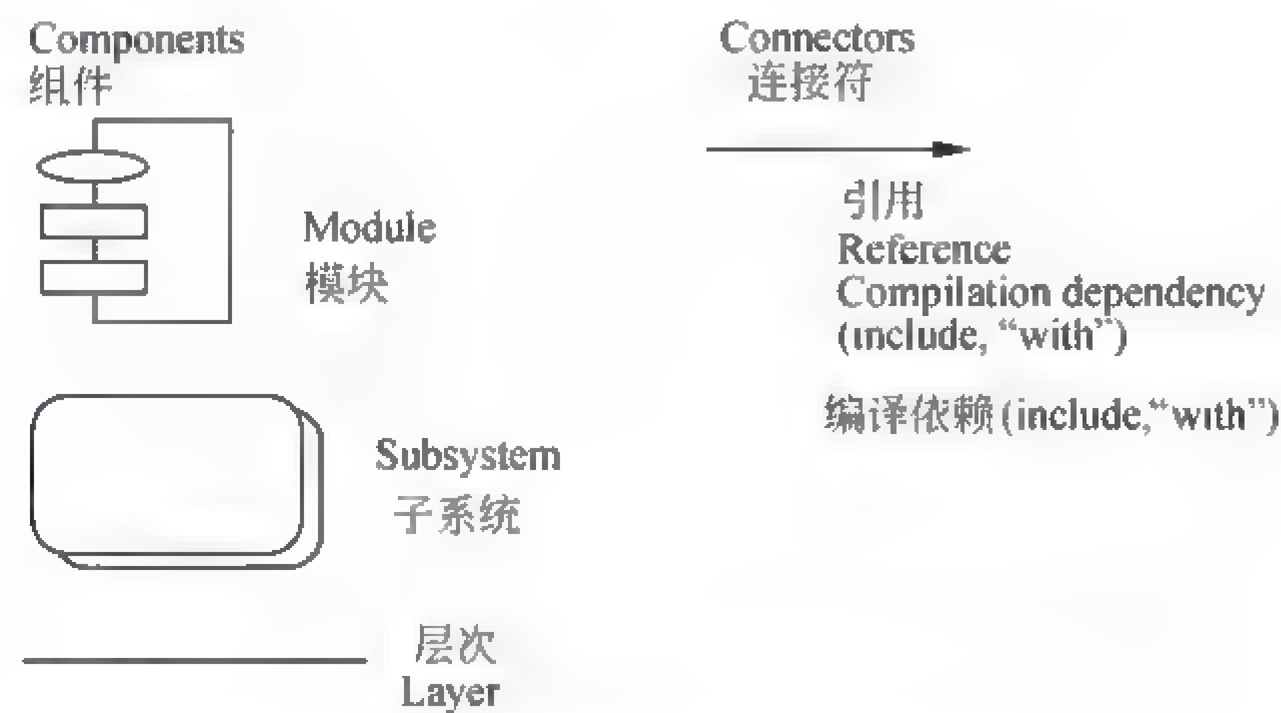

图 2-10　面向 Booch 的开发视图标记法

2.1.3.4　物理视图之硬件到软件间的映射

物理视图主要关注系统的可用性、可靠性，性能（吞吐量）和可伸缩性等非功能性。而软件是运行在计算机网络或处理结点上，且被用于识别网络、过程、任务和对象的元素，但需要将网络、过程、任务和对象等元素映射到硬件的不同结点和处理点上。同时，由于不同的物理配置、开发、测试方式在不同地点进行部署，这就要求软件映射具有高度的灵活性。图 2-11 是面向 Booch 的物理视图标记法。

2.1.3.5　场景

场景主要作用是协同其他四种视图工作，也是其他四种视图相互转换的重要需求抽象。同时，也是作为一项驱动因素来发现架构设计过程中的架构元素和架构设计结束后的一项验证和说明功能。而场景的表现方法与组件逻辑视图非常相似，且使用过程视图的连接符来表

示对象之间的交互，特别地对象实例使用实线来表达。

图 2-11　面向 Booch 的物理视图标记法

2.1.3.6　各视图关系与开发设计

各种视图并不是完全正交的或独立的，而是视图的元素根据某种设计规则和启发式方法与其他视图中的元素相关联。从而使各视图连接成一个整体，形成一个可操作、可建模的软件架构设计。

1．逻辑视图与过程视图关系

通常情况下，在逻辑视图中的对象是主动的，所表现的仅是需求的功能性，而且逻辑视图具备自主性、持久化、依赖性和分布性等特征，使得逻辑视图具有潜在的并发性，就需要以某种抽象的方式来调用相关的操作；这时，一般采用"从由至外"和"从外至内"的策略来决定并发的程度和定义所需的过程集合。其目的是将对象（类）映射到一个任务集合和进程架构中的进程去，其中的活动类具有代理任务，也存在一些变形：对于给定的类，使用多个代理以提高吞吐量，或者多个类映射至单个代理，因为它们的操作并不是频繁地被调用，或者是为了保证执行序列；但这并不是产生最佳过程架构的决定性和线性的进程，而是需要若干个迭代来得到可接受的折衷。

2．逻辑视图与开发视图关系

类通常作为一个模块来实现，并通过创立对象来实现应用，使得密切相关的类（类的种类）的集合组合到子系统中，形成一段有序且能实现某功能的程序。但子系统的定义必须考虑额外的约束，如团队组织、期望的代码规模、可重用性和通用性的程度以及严格的分层依据（可视性问题），发布策略和配置管理等。因此，通常得到的不是与逻辑视图逐一对应的视图。而逻辑视图和开发视图非常接近，但具有不同的关注点。通常情况下项目规模越大，视图间的差距也越大。

3．进程视图与物理视图关系

进程和进程组以不同的测试和部署配置映射至可用的物理硬件上，使得程序可以正常运行并能读取所需的数据。并以场景将所使用类的形式去与逻辑视图相关联；而与进程视图的关联则是考虑了一个或多个控制线程的、对象间的交互形式。

但并不是所有的软件架构都需要"4+1"视图，这时就需要考虑视图模型裁剪，将无用的视图从架构描述中省略。而在具体应用"4+1"视图进行软件架构时，需要将整个架构过

程用文档的形式记录下来，其目的是保持软件架构的可操作性、可测量性和可设计性。这些文档一般包括软件架构文档和软件设计准则。

2.1.3.7　场景驱动（scenario-driven）的方法

系统大多数关键的功能是以场景或用户案例的形式被捕获，并根据场景变化和用户的需求制定捕获业务功能的策略，使其在不同的阶段实现迭代来实现软件设计和架构，其步骤如下：

1．开始阶段。有以下五步

（1）由于场景可能被归纳为对若干用户需求的抽象，所以基于风险和重要性需要为某次迭代选择场景。

（2）形成“稻草人式的架构”，然后对场景进行描述，以识别主要的抽象，一般包括类、机制、过程、子系统等。

（3）将所发现的架构元素分布到逻辑视图、进程视图、开发视图和物理视图中。

（4）然后实施、测试、度量该架构；这步分析可能检测到一些缺点或潜在的增强要求。

（5）捕获经验教训。

2．循环阶段

首先进行下一个迭代准备：

（1）重新评估风险。

（2）扩展考虑的场景情况。

（3）选择能减轻风险或能提高结构覆盖的额外的少量场景。

再进行迭代的具体操作：

（1）试着在原先的架构中描述这些场景。

（2）发现额外的架构元素，或有时还需要找出适应这些场景所需的重要架构变更。

（3）更新逻辑视图、进程视图、开发视图和物理视图。

（4）根据变更修订现有的场景。

（5）升级实现工具（架构原型）以来支持新的、扩展了的场景集合。

（6）测试，如果可能的话，在实际的目标环境和负载下进行测试。

（7）然后评审这五个视图来检测简洁性、可重用性和通用性所存在的潜在问题。

（8）更新设计准则和基本原理。

（9）捕获经验教训。

3．终止迭代

但针对不同的项目，所迭代的时间是不一样的，这是因为受到所实施项目的规模，参与项目人员的数量、对本领域和方法的熟悉程度影响，以及该系统和开发组织的熟悉程度等因素的影响。

2.1.4　软件构架师

软件构架师可以比作是项目的导演，他并保证所研发的软件产品能符合软件企业等机构的要求。因此，软件构架师是一个技术主管，这就表明构架师不但要掌握全面、丰富的技能外，还应该具备领导、指导才能；并且软件构架师的领导能力在整个团队中和项目质量控制

中具有主要的作用。同时，在整个团队中，软件构架师就是项目的技术总管，需要有丰富的知识和经验，以便做出技术上的决定，以及能在整个构架小组中起到推动作用；并且能以敏锐的感知能力聚焦各方智慧为项目顺利实现提供可靠的指导和帮助。而相对于项目经理来说，项目经理是来管理项目的资源、时间进度和花费的，只对整个项目宏观进行管理，即对资源分配、时间和花费提供均衡管理，因此，可以将项目经理比作是项目的“制片人”。由于软件构架师和项目经理在项目中所处的位置不同和所具备的职能不同，构架师和项目经理是公众人物，在一个团队中，他们是整个项目所涉及的所有人员的联系枢纽；因此：

（1）构架师应该为建立软件构架争取投资，并且要明确建立软件构架能给组织带来的价值；还要把团队组织在构架周围，并且要积极地投入到计划活动上，因为要把构架转化成为完成任务的先后顺序，这样才能及时地确定在什么位置需要什么技术。

（2）构架师需要根据具体的实例情况来做领导决定，并且在决定过程中要展现出足够的自信。

（3）构架师还要把精力放在切实工作的交付上，因为他是技术方面的推进力量；同时，构架师需要做决定，并且要保证这些决定是经过成员之间的交流的，并且确保它能够执行。

下面从构架师的所具体的技术和领导能力两个方面概述作为导演的软件构架师作用。

2.1.4.1 软件构架师技术能力

软件构架师的技术能力主要从以下方面来进行概述，从而使软件构架师满足所具备的基本技术能力。

（1）软件构架师应该理解整个项目的软件开发过程：构架师能对软件开发过程有正确的估计，因为这个过程可以确保小组中的所有成员使用同等的方式工作；一个好的过程需要定义各个角色的工作承担责任、产品的建立导向、不同角色之间的协同工作等，并能明确自己的职责，了解该做什么工作，以及怎么去做，以及在项目小组中能为软件设计师、程序员提供指导和帮助。

（2）软件构架师需要有商业领域的知识：软件构架师需要从商业需求角度权衡软件开发的价值和在不同领域应用能力，以及后续生命周期。因此，一个好的构架师将会在软件开发和商业领域的知识上面做出权衡。如果一个构架师具有很好的软件开发经验但是不了解商业领域，那么他的解决方案可能不会解决实际的问题，也不能被需求所采纳，而仅仅只能反映出构架师是多么精通他的专业，这样就达不到对软件构架师的要求。同时，软件构架师还应具备预见软件构架随时可能出现的变化，对不断变化的情况做出更有远见的决策。

（3）软件构架师应该拥有技术知识：软件构架的一个特定方面需要较丰富的专业知识，因此一个构架师必须具备这个水平的知识才能够胜任他的工作，可是构架师不必成为技术专家，但构架师必须了解技术宏观上的问题，而不必关心技术细节化的事情，但技术的变化过于频繁，所以构架师要随时与这些变化保持同步。

（4）软件构架师应该具备很好的设计技巧：软件构架并不仅仅是设计，还是整个项目的技术指导者，但对构架师来说，不是为了设计而设计，而应拥有很好的设计技巧，因为软件的构架包含整个软件的关键性设计决策（一般包括软件主要结构的设计决策，特定部分的选择以及指导的说明文档等），以及是后续实现的关键，还是有效降低实现成本的一个重要因素之一。为了确保系统构架的完整性和有效性，这些因素都要被特别的应用到设计中，这对整

个系统的成功完成有很大的作用。

（5）软件构架师应该具备好的程序设计技巧：程序开发人员是实现项目最重要组成部分。而构架师要与开发人员保持密切的联系，以便及时掌握项目开发所表现出来的不确定因素和不可抗拒的问题，因此，构架师需要及时、有效的解决这些因素和问题。这就要求构架师需要拥有一定的程序设计技术和程序开发的丰富经验，这时不需要亲自动手编程。但往往很多成功的构架师，在一些项目开发中都是核心程序员，这些场合通常是他们的职业方向的要求。即使是技术发展了，有新的程序语言出现，一个好的构架师可以把以前学过的设计语言的概念和新的语言联系起来，以达到对新语言更加深入的了解。

2.1.4.2 软件构架师领导能力

软件构架师的领导能力主要从以下四个方面进行概述，从而明白作为一名构架所具体的基本领导能力。

（1）构架师是一个很好的沟通员和联络员：同样软件构架师需要具备沟通和联络能力，使得他是一个很好的聆听者、观察者和指导者。因为项目团队中成员之间有效的沟通是项目成功的基本条件，也是构架师领导能力的突出表现。主要表现在与投资者（制片人）和小组成员间的沟通和联络，使得投资人有信心并能在软件构架上达到共识；而对团队成员间的不仅表现在沟通和联络上，更重要的是激励他们工作，肯定他们的工作，即用委婉的方式指导他们工作，让他们在快乐、和谐的沟通中创造价值。

（2）构架师需要对项目实施做出决策：加拿大著名软件构造师 Philippe Kruchten[①]说：软件构架师的一生是一个漫长的，在黑暗中不断摸索并不断改进自己的决定的过程。这就表明构架师的决策是整个项目的关键。因此，构架师不能在自己不了解的环境中做出决策，不能草率的对整个项目做出有影响的评论，需要有充足的时间去探索与项目相关的环境，然后再根据具体情况一步一步的做出决策。这是因为一个不成熟的决策很可能毁掉一个项目，相应项目小组中的其他成员也会对构架师失去信心，使得失去构架师的作用。

（3）软件构架师需要觉察组织的政策：一个成功的构架师不会只关心技术问题，他们还会关心组织的权力动向，时刻了解团队的决定权在哪里，以便掌握软件构架的主动权，掌握项目技术走向；这不但是对投资者负责，也是对团队中其他成员负责，也可以保证他们正确的讨论项目的决策问题。然而忽略团队的权力是天真的想法，也是不可取的，而现实使得团队经常会强迫项目小组在规定时间交付系统，这需要构架师正确的评估到这个时间。所以作为构架师需要动态掌握整个项目的权力动向。

（4）软件构架师是一个谈判代表和技术领导：为了了解软件构架的很多尺度性问题，构架师需要随时和投资人沟通，这种沟通常常需要谈判技巧，也需要随时与团队成员进行指导和激励。例如，构架师需要特别注意的一件事是：最小化项目中可能出现的风险，因为这直接关系到系统构架的稳定性。同时，由于风险是和需求紧密相连的，所以可以通过移除或者减小这方面的需求来降低风险，因此把这种需求取消，需要构架师和投资人达成共识的。这就需要构架师是一个有效的谈判人员，来权衡这些问题。以及保持与团队成员间的亲密、和谐关系。

① http://en.wikipedia.org/wiki/Philippe_Kruchten

2.1.5 案例分析——档案管理系统

某项目档案管理系统的需求是实现项目归档，即将已完成的项目进行归档处理，将正建项目实现跟踪管理，以及将未完成的、特殊的项目进行备案管理，并能按要求将各项目的状态用报表的形式进行输出。而系统功能是指在不同的角色的权限下，实现项目档案管理，即不同的角色可以操作不同的功能，其系统功能包括四部分，一是项目主管审批部门使用的功能，例如发改委；二是项目申报业主使用的功能；三是报表输出功能；四是强大的系统管理功能。下面根据软件构架基本方法和思想分析某项目档案管理系统。

1．项目档案管理系统网络拓扑结构

如图2-12是项目档案管理系统的网络拓扑结构立体图，该图展示了系统的网络结构，以及满足的主要操作人群的终端，当然随着通信的深入应用，还能引入无线网络融入到该体系结构中。其简化的项目档案管理系统网络拓扑如图2-13所示。

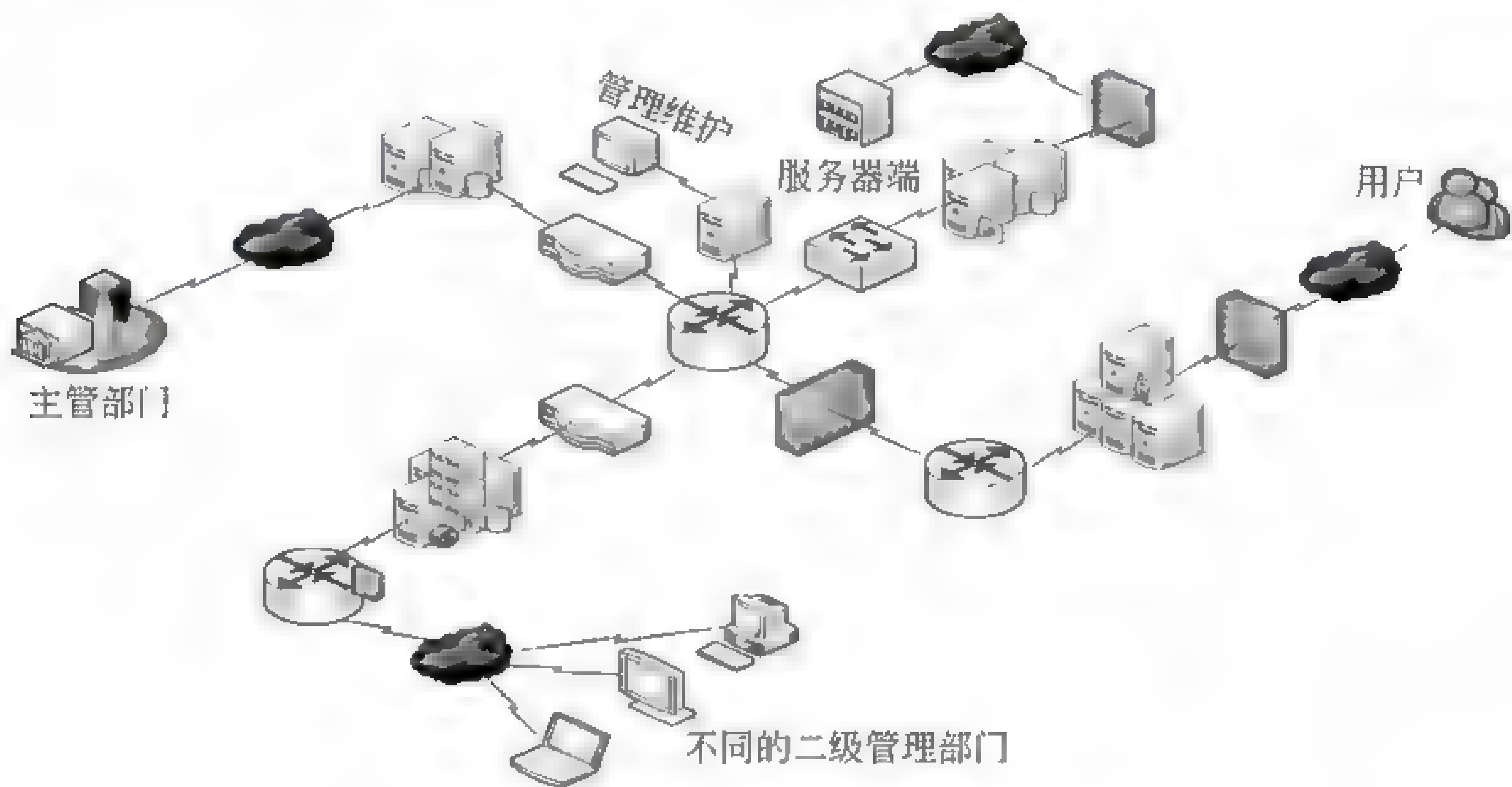

图2-12　某项目档案管理系统网络拓扑图

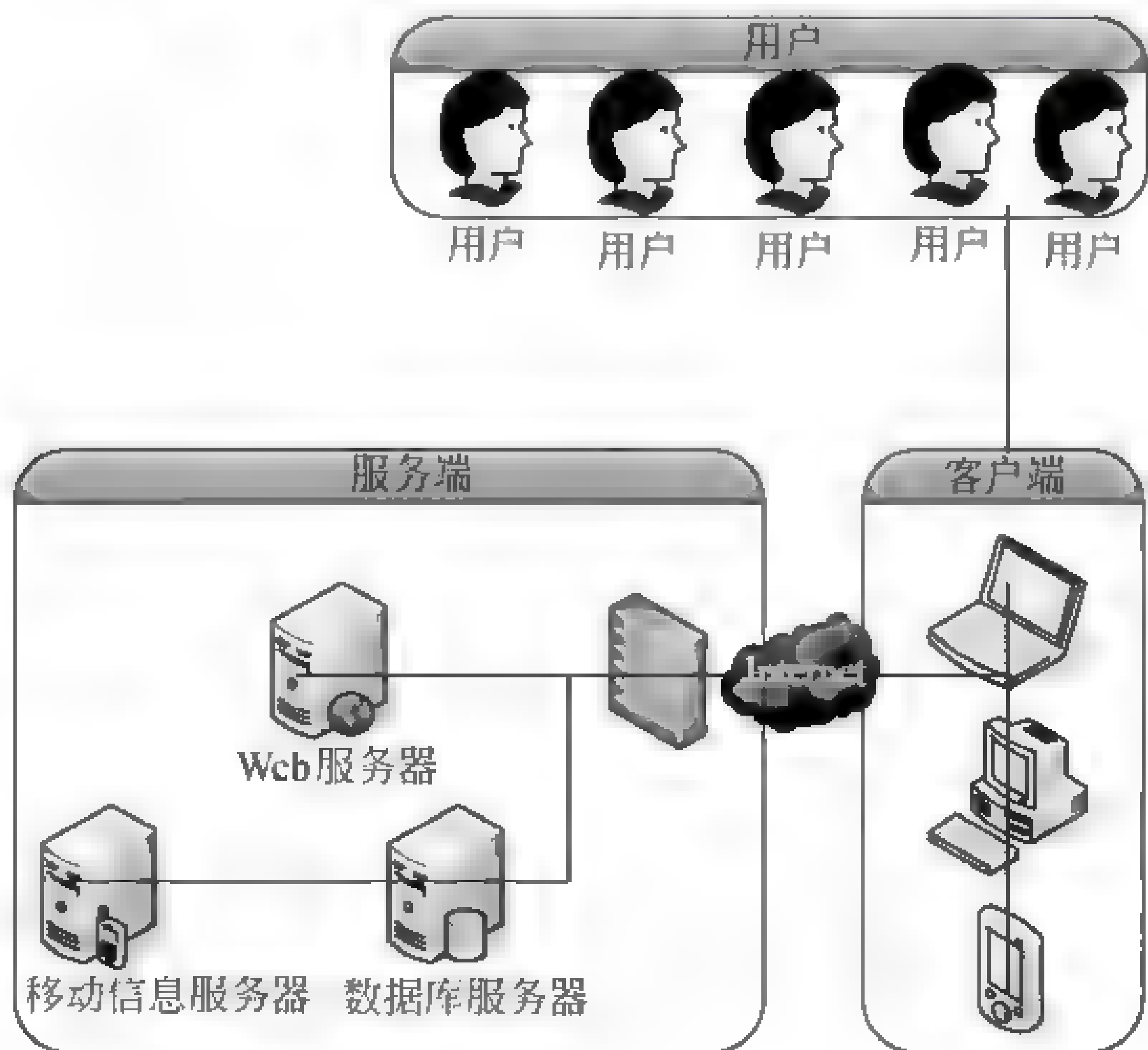

图2-13　简化后的某项目档案管理系统网络拓扑图

2．项目档案管理系统逻辑结构

主要由表示层、业务逻辑层、数据逻辑层和数据管理层组成，如图 2-14 所示。

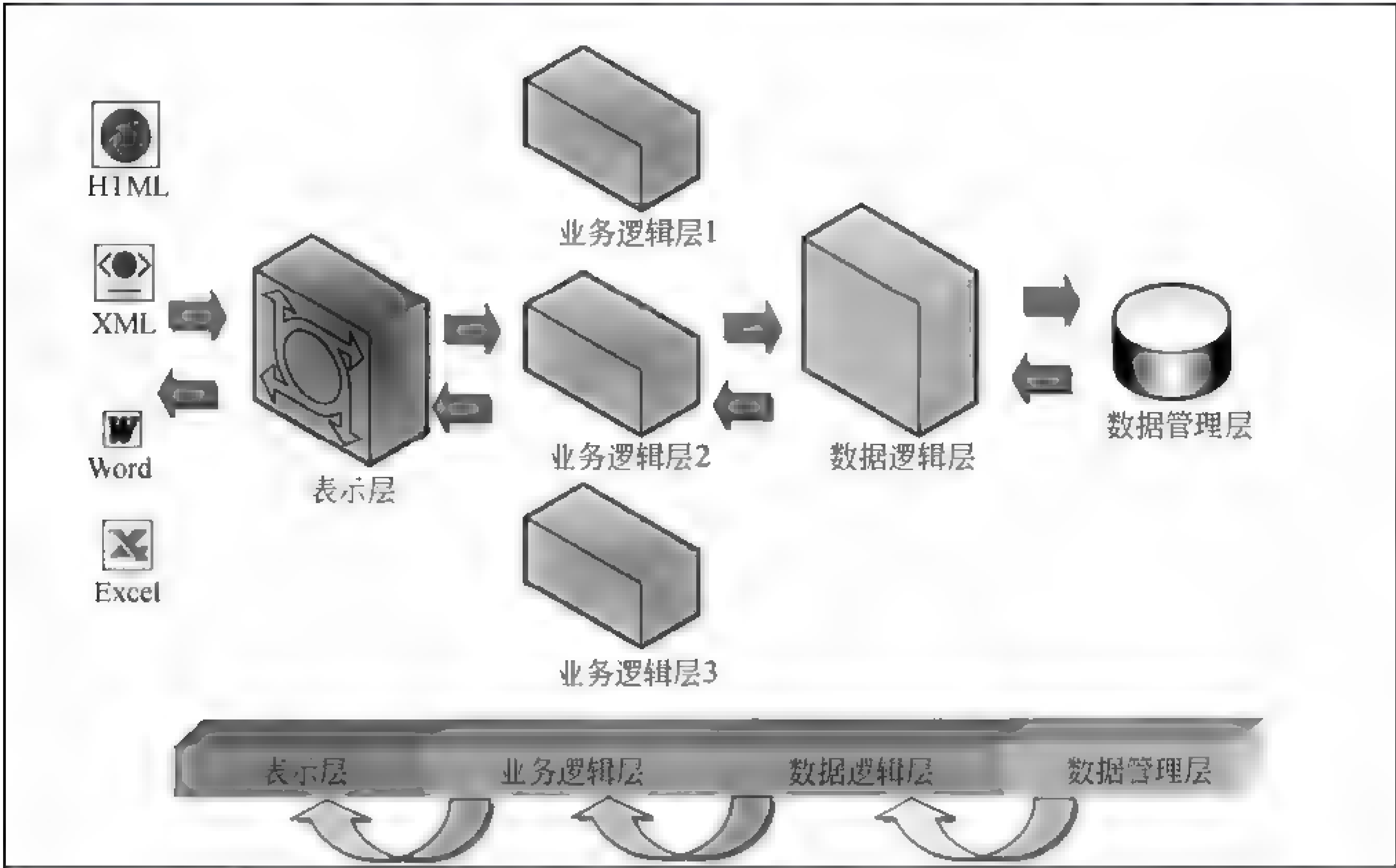

图 2-14　某项目档案管理系统逻辑结构图

表示层：表示层将系统的数据通过网页、Word、Excel、门户等多种表现形式展现给客户，并实现与客户的交互。

业务逻辑层：业务逻辑层是系统架构中体现核心价值的部分，它的关注点主要集中在业务规则的制定、业务流程的实现等业务需求有关的系统的设计，也就是说它与系统对应的领域逻辑有关。业务逻辑层在体系结构中的位置很关键，它处于数据逻辑层与数据层之间，起到数据交换的承上启下的作用。

数据逻辑层：数据逻辑层又称数据访问层，它将数据的表示逻辑和数据的访问逻辑清楚地分开。

数据管理层：数据层作为信息的载体，对逻辑应用提供的开发的标准接口，并且能够保证数据的可靠性、可用性和安全性。

3．项目档案管理系统体系结构

图 2-15 展示了整个系统的体系结构及组成，全面显示了项目管理档案系统组成结构及操作人群（角色）。

4．项目档案管理系统流程

图 2-16 是某档案管理系统流程图，展示了整个系统业务流程的流向及业务功能的构成。

5．项目档案管理系统报表输出

图 2-17 是项目档案管理系统报表输出结构图，主要由报表读取的数据库、报表和操作界面组成。

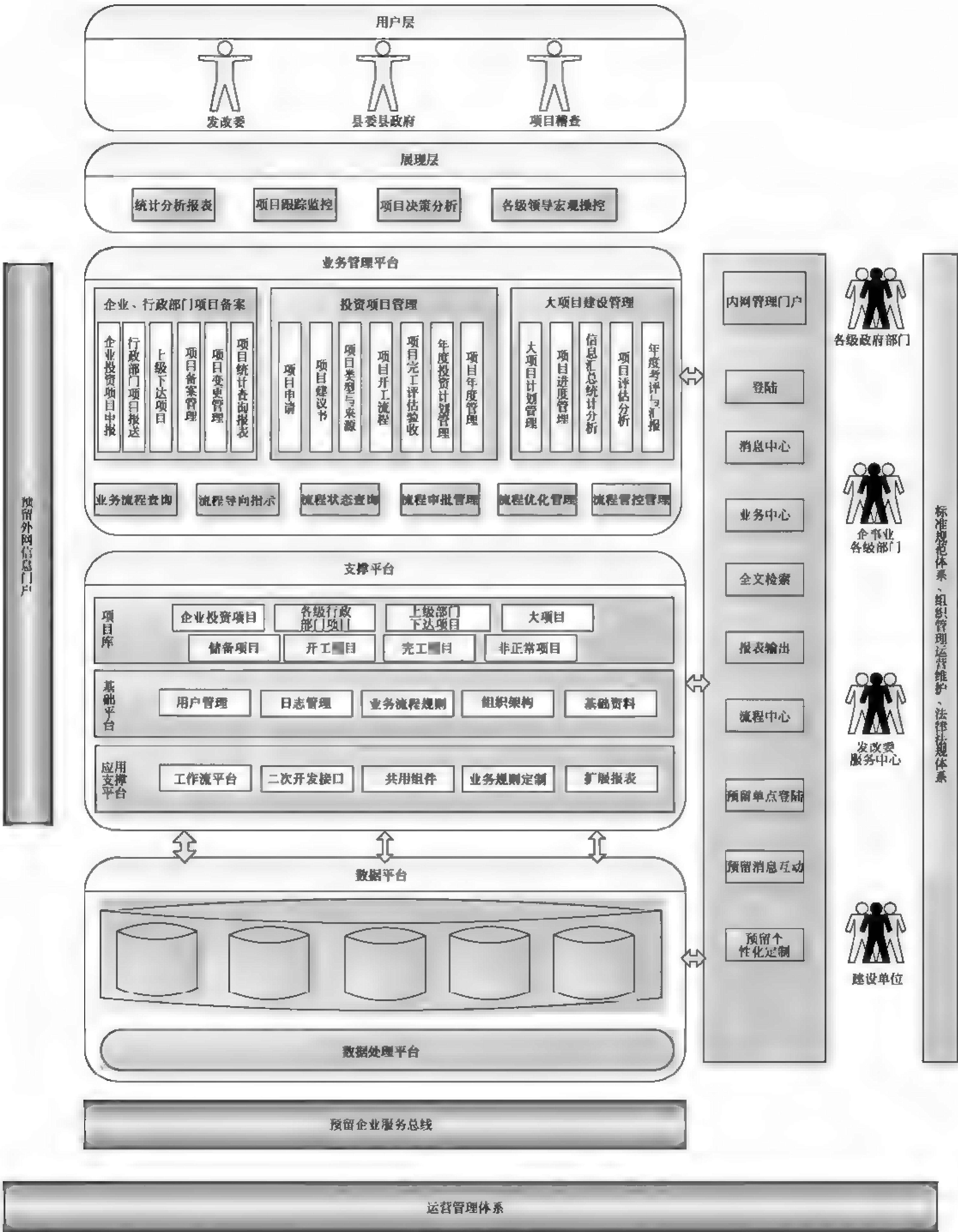

图 2-15　某项目档案管理系统体系结构图

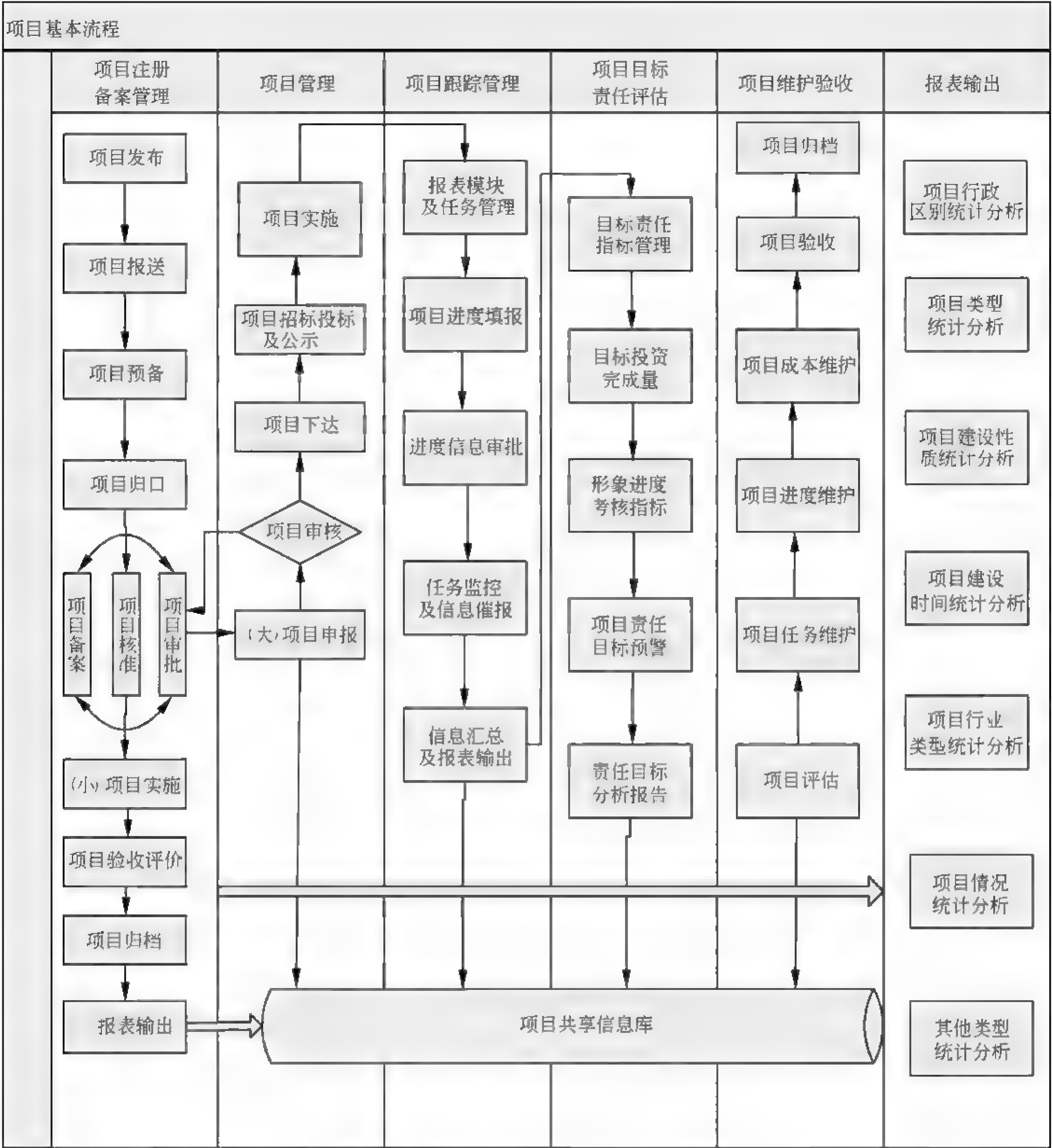

图 2-16　某项目档案管理系统业务流程图

在图 2-17 中的报表中的功能介绍如下：

文件接收：接收各系统下传的报表文件。

报表生成、转换：负责生成带格式的报表文件，并将其转换成 PDF 文件，同时将文件名加密。

权限设置：设置报表输出权限。

报表分发：设置报名分发功能，包括报表的派发，按机构的查询/打印下载等功能。

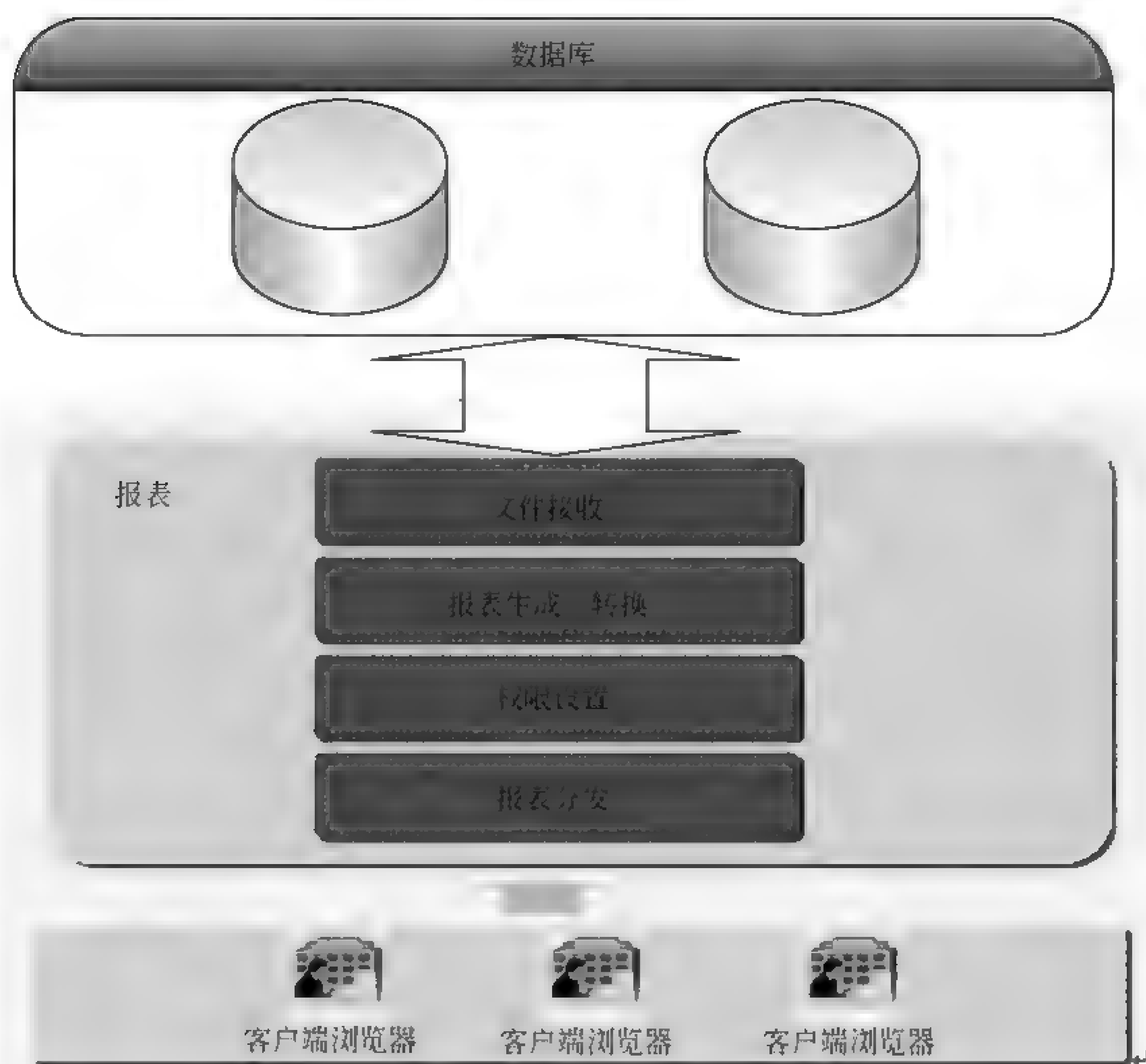

图 2-17 某项目档案管理系统报表输出结构图

2.2 基本的软件构架方法

在 2.1 节从宏观上概述了软件构架的概念和基本的方法，包括软件构架的质量评估、“4+1”视图模型、软件构架师，并以一个具体案例进行了分析。这些基本方法为软件构架提供了指导思想，为实现具体的软件提供了蓝图和对事物的抽象。若将这些软件构架基础应用到具体的需求进行建模，就可以用相关工具来实现该软件模型，但这还不能进行详细的分析设计出一套合理、满足需求各方要求的软件系统，这是因为软件构架是对软件系统整体组织结构和控制结构的刻画，包括系统中各计算单元（构件）的功能分配、各单元之间的高层交互说明（连接件）以及 SA（Software Architecture）的约束。同时，根据 1999 年 IEEE 的定义，将软件体系结构定义为软件系统的高层抽象，描述整个系统的结构和行为模型，主要由构件、连接件及将构件和连接件结合在一起的约束条件构成。

软件体系结构已成为软件工程领域的一个较新的研究热点，并在各个领域的软件构架中起到推动作用。软件体系结构主要是研究构件，构件之间的交互作用、全局控制结构、通信协议、同步协议和数据访问等。简单的说，就是研究构件及构件之间的交互，以及研究抽象软件体系结构的共性，即构件和连接。而构件主要关注构件的分离点和层次粒度重用，它的目的是避免构件的数据结构和算法选择变得更为初始和宏观。然而，软件体系结构模型是描

述某一特定应用领域中系统组织方式的惯用模型，它反映了领域中众多系统所共有的结构和语义，并指导如何将各个模块和子系统有效地组织成一个完整的系统。

按照 Garlan & Shaw 的分类，可以将软件体系结构分为：数据流模型、调用\返回模型、虚拟机模型和仓库模型[10]。而在 2000 年第十六届世界计算机大会中，著名软件体系结构专家 perry 指出，在 SA 中最为重要的三个研究方向即体系结构风格、体系结构连接件和 DSA（Dynamic Software Architecture），这为软件体系结构的研究和应用提出了新的指导方向，使得众多研究就以此为依据开展多样化的研究；但对软件体系结构的描述和表达主要有两类[11]：一是形式化描述方法，这种方法以形式化技术作为语义信息的理论基础，如π-calculus、偏序事件集理论、CSP 以及 Petri Net 等；形式化描述方法的抽象程度高，采用专用符号，难于被开发人员理解，不便于交流和使用，容易使得状态空间急剧增长，缺少工具支持，且较难融入到当前软件开发的实践中。二是以统一建模语言 UML 为代表的可视化描述方法，是事实上的工业标准，可以从多个视图来描述软件体系结构，但存在语义不完整的缺点。

2.2.1　软件体系结构论述

根据现有的软件体系结构研究状态来看，软件体系结构主要从概述（包括现状、发展等）、构件的状况及形式化描述方法、软件体系结构描述方式、软件体系结构风格、体系结构方法、软件体系结构基本应用和软件体系结构可靠性分析方法等角度展开软件体系结构研究。而在本书中是将现有的软件体系结构的思想、方法应用去指导面向开源软件的软件开发；当然这些内容是可以完全应用到开源软件中去，这是因为开源软件不是一种新的软件体系结构方法，只是一种软件发展的形式，以及是一种以开源的形式向软件研究、工程人员提供一种更为自由的操作模式，更是一种聚焦共同智慧推动软件快速发展的有效策略。同时，开源软件的体系构架是与传统的体系构架是基本相同的，这是因为开源软件仅是一种软件研发模式。但目前很少涉及到从软件体系结构、构架模式、需求分析等角度展开分析研究，这是由于普遍认为开源软件只是一种软件研发方法；其实随着开源软件和网络发展，以及形式多样化、展现复杂化，各种软件形态逐渐呈现于网络中的，单纯的借鉴传统的软件构架模式来指导面向开源软件的软件开发，会逐渐呈现出困难。

2.2.1.1　软件体系结构研究挑战与进展

可以这样认为，软件体系结构是由 20 世纪 60 年代时的软件危机催生而来的，然而到 1995 年出版的 IEEE Software 体系结构专刊和 1996 年出版的专著《Software Architecture: Perspectives on an EmergingDiscipline》才可以认为是 SA 作为软件工程一个研究方向正式提出的标志。起初人们重点将软件设计放在数据结构和算法选择上，但随着软件系统规模越来越大、需求越来越复杂、整个系统结构和规格说明越来越重要，也越来越突出，因此就需要以工程化的思想来指导软件构架，这时人们深入研究软件体系结构后认为软件体系结构是提高软件生产率和降低软件成本最有途径之一。而软件危机主要是在软件成本日益增长、开发进度难以控制、软件质量越来越差、软件生命周期越来越短、软件维护困难等情况下产生的，造成这些的原因是软件用户需求不明确、缺乏正确的软件理论指导、软件规模越来越难控制、软件复杂度越来越高、程序冗余度越来越大、软件集成性不高、兼容性不确定等因素引起的。

在此背景下，软件体系结构就逐渐兴起，它是作为控制软件复杂性、提高软件系统质量、支持软件开发和复用的重要手段之一，因此成功指导了软件的构架和研发，基本上解决了软件危机所带来的问题，也改变了程序设计的语言的结构和软件开发模式，如由最初的面向机器的语言设计，到面向过程化的程序语言，到面向对象的程序语言，再到面向构件的软件开发，以及近年来的面向服务、面向开源软件的软件开发等，但不管是程序设计语言的发展，还是软件开发模式的更新，它们的共同的目标提高软件可重用性、增强软件的灵活性、提高软件松散耦合度、提升软件层次粒度等。并且在软件需求越来越复杂、软件工程越来越多样化的网络化的软件构架时代，怎样将提高软件开发效率，降低软件成本、增强网络化的软件重用率等，仍是一个软件构架的难点和重点。这时，软件工程加上需求工程，以及面向网络化的软件重构就产生，并用来指导新型网络化软件的构架。

在过去很长一段时间内，众多软件系统的研发，是以功能变化来实现软件构架和研发，这就使得在软件结构网络化、多需求复杂化、可重用的软件实体不确定化等影响下，给软件重用率带来的前所未有的挑战。但随着网络化软件的发展，以需求为中心的动态演化的软件构架思想逐渐形成，在一定程度上有效解决了这种困难，并指导软件构架和研发向面向服务、面向服务体系结构、面向云等新型软件体系结构方向发展。特别地，作为面向开源软件的软件开发的新型软件研发模式是实现网络化（Internet）软件发展的创新，因此，怎样将面向开源软件的软件开发形成一门软件构架的新型模式，以及软件研发的新型方法是目前摆在人们面前又一难题。这时因为开源软件有别于构件、有别于其他任何软件形式，它是一个一个的软件实体，是面向所有需要的用户提供开源，而且涉及到的面广，到目前为止，开源软件几乎的涉及到了所有软件领域，这就要求人们用一种适应开源软件的构架体系结构和研发模式来指导面向开源软件的软件开发，从而使开源软件在软件构架和研发过程具备集成性、开放性、复用性、服务性等多重优势。图 2-18 所示是基于软件体系结构的软件开发过程[16]。

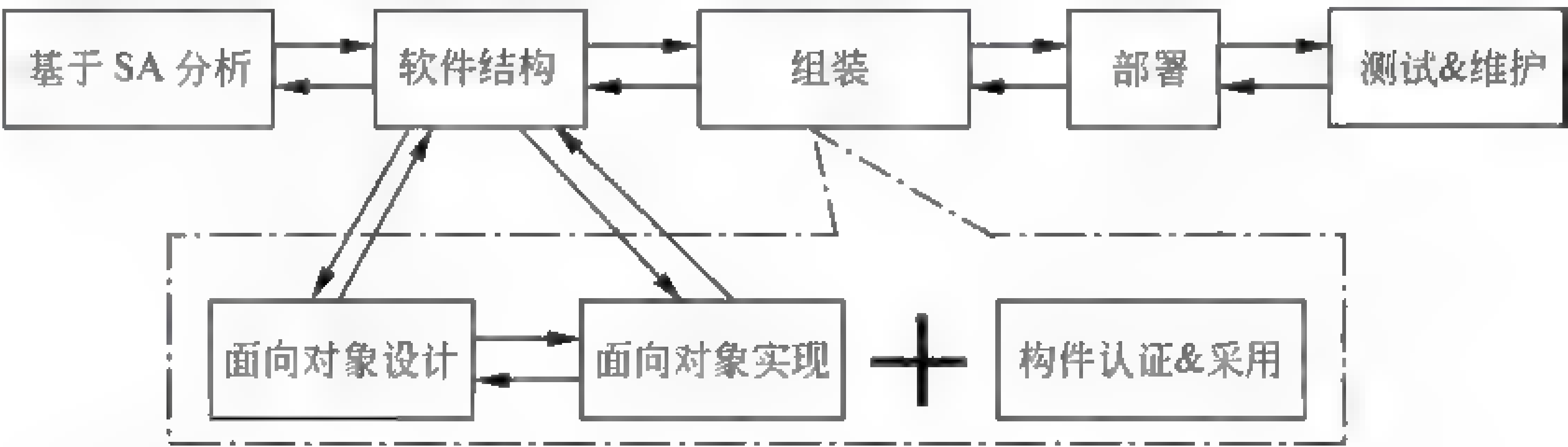

图 2-18　基于软件体系结构的软件开发过程

1．软件构架面临的挑战

根据 IDC（Internet Data Center）的研究，未来 5 年，企业的结构化数据每年会保持 20% 的增长，而非结构化数据量的增加则高达 60%。数据量急剧增加的一个必然后果是软件越来越大，也越来越复杂，需求也越来越多样化，直接给软件构架带来了挑战，即怎样在网络化环境中合理利用这些数据，并为软件构架服务，也能为面向网络化的软件研发提高可重用率、控制软件的复杂度和提高软件质量。同时，这些越来越多的海量数据怎样支持更复杂的数据源，更为灵活的软件自动配置、演化和重组呢？这就要求最终需要一种更灵活和可靠的软件架构来实现，而传统的软件架构显然难以适应新的 IT 环境，特别难以实现网络化软件的 IT

环境。

在过去，IT（Information Technology）中的软件技术主要用于完成一些可重复的业务流程的自动化，且让主要业务流程为软件构架的对象，如 ERP（Enterprise Resource Planning）应该是目前企事业 IT 最典型的应用方式之一，它实现了订单、记账和库存跟踪的自动化，从而大幅度提高了企事业的工作效率。而如今 IT 技术的这种应用方式正在发生变化：即越来越难以满足用户需求，也难以自动的处理各方面要求；因为今天的企业所经营的业务已经很难和 IT 分开，这时需求用一种的新模式来处理日益增长的数据量、业务量，即使用目前的商业智能（Business Intelligence）将业务、需求、数据三者分开。但应用程序往往需要处理的数据量、数据类型以及应用程序是随应用场景的变化而变化，这样就给未来的应用软件构架带来很多挑战，主要包括：

（1）应用程序变得复杂且负载变化大：大量的数据的存在，给应用程序的负载变化增大，主要表现在工作任务变化和工作环境的复杂都使得应用程序的负载变得不可预测，也难以控制。这直接给面向网络化的软件构架带了挑战，即怎样使用一种软件构架方法来实现需求分析、业务聚焦和数据分析与处理，从而满足不同类型用户的使用要求。

（2）软件实体类型更加复杂且分布更广泛：由于分布于网络中的软件实体类型多且分布非常广泛，怎样将这些可以使用的软件实体自动聚集起来为软件构架服务是当前面临的挑战。这些软件实体可以进一步增强软件可复用性，实现软件构架、研发能自动在网络中实现，即通过需求来实现软件自动生成。

（3）应用程序的拓扑结构更复杂：由于数据处理规模的不可预测，就要求软件架构的设计必须改变。比如，很多程序开始大量使用内存缓存数据以提高处理速度，而不一定把每个都保存到数据库。而且，应用程序复杂常常是与异步处理和计算密集性的任务相伴相随的，这时通常都会用到消息队列，在应用程序的软件架构时都必须考虑到这些问题。

（4）新技术的出现：近年来，兴起的服务计算、云计算和物联网技术等对解决日益扩张的数据处理与分析提高了软件构架方法。服务计算通过将业务实体、软件实体等以服务的形式发布，然后供人们选择和使用，并在一定程度上实现服务组合完成不同的服务请求者的需求。云计算更是一种高伸缩性的计算模式，为网络化的软件构架提供了可操作性。而物联网的兴起更是推动了应用软件的变化，使得软件从人机交互为主到全面的自动化。

在参考文献[12]中提出一种以体系结构为驱动的移动框架，并指出许多软件体系结构是受软件生命挑战。包括波动的运行环境（Fluctuating execution context）、资源有限（Constrained resources）、异质性（Heterogeneity）、特有基础设施（特有的基础设施）、实现效率（Efficiency of the implementation）、代码异质性（Coping with heterogeneity）、灵活性和可扩展性（灵活性和可扩展性）、环境认识（Context awareness）和体系结构的支持（Context awareness）等。

2．软件体系结构研究进展

下面就对软件体系结构研究进展进行概述，包括软件体系结构的研究方向、研究进展和存在的不足[13]。其主要表现如下：

（1）软件体系结构的研究方向：

① 体系结构理论模型的研究；

② 体系结构描述研究，主要研究体系结构描述语言及其支持环境、体系结构描述规范；

③ 体系结构设计研究：包括体系结构设计方法、体系结构风格、体系结构设计空间等

内容；

④ 体系结构分析与验证：研究如何将软件的非功能特性转化为体系结构的需求，如何分析体系结构满足期望的需求属性，对体系结构的语法、语义、类型失配等进行检查与验证的研究；

⑤ 体系结构演化与复用研究：研究产品线中软件体系结构演进的理论与方法，从已有文档、系统设计和代码中逆向提取软件体系结构、体系结构复用等；

⑥ 动态体系结构研究：研究软件系统由于特殊需要必须在连续运营情况下的体系结构变化与支撑平台；

⑦ 基于体系结构的软件开发：研究引入体系结构后的软件开发过程、基于体系结构开发与中间技术集成、基于体系结构的程序框架自动生成技术等。

（2）软件体系结构的研究进展：

① 软件体系结构描述语言 ADL（Architecture Description Language）：在 2002 年以前就已经提出了若干 ADL，主要包括一种基于构件和消息的 ADL-C2，它适用于大型频繁交互的层次型图形用户界面的软件的体系结构描述；支持从一种 ADL 向另一种 ADL 规格转换的体系结构互换语言 ACME；基于框架和角色模型的软件体系结构规约 FRADL；多智能体系统体系结构描述语言 A-ADL；可视化体系结构描述语言 XYX/ADL；基于主动连接件的体系结构描述语言 Tracer；等等。

② 体系结构描述构造与表示：主要包括①Kruchten 提出的“4+1”模型；②Booch 提出从 UML 的角度给出了一种由设计视图、过程视图、实现视图和布署视图，再加上一个用例视图构成的体系结构描述模型；③1995 年由 IEEE 起草的体系结构描述框架标准 IEEE P1471；④Rational 从资产复用的角度提出了体系结构描述的规格说明框架（Architectural Description Ecification），等等。

③ 软件体系结构分析、设计与验证：体系结构分析的内容可分为结构分析、功能分析和非功能分析。其中在进行非功能分析时，可以采用定量分析方法与推断的分析方法，这些方法主要包括有 Kazman 等人提出的 SAAM 等；而在非功能分析的途径上，则可以采用单个体系结构分析与体系结构比较的分析方法。

生成一个满足软件需求的体系结构的过程即为体系结构设计。体系结构设计过程的本质在于将系统分解成相应的组成成分（如构件、连接件），并将这些成分重新组装成一个系统。具体说来，体系结构设计有两大类方法：过程驱动方法和检查列表驱动方法。

④ 软件体系结构发现、演化与复用：软件体系结构发现解决如何从已经存在的系统中提取软件的体系结构，它属于逆向工程范畴。系统需求、技术、环境、分布等因素的变化而最终导致软件体系结构的变动，称之为软件体系结构演化。而体系结构复用属于设计复用，它比代码复用更抽象，一般认为易于复用的标准包括领域易于理解，变化相对慢，内部有构件标准，与已存在的基础设施兼容，在大规模系统开发时体现规模效应；因此，复用技术作为软件工程领域倡导的有效技术之一，在基于构件与体系结构的软件开发时代，软件体系结构复用将是一个重要的主题。

⑤ 基于体系结构的软件开发方法研究：软件体系结构是对软件需求的一种抽象解决方案，是在软件需求与软件设计之间的一座桥梁，即问题定义→软件需求→软件体系结构→软件设计→软件实现。而在由软件体系结构到实现的过程中，一般借助一定的中间件技术与软

件总线技术（如服务总线技术），软件体系结构将易于映射成相应的实现。

目前，基于构件和基于体系结构的软件开发已成为主流的开发方法，并且在各领域得到了很好的应用（如北大青鸟构件库、上海普元构件库等），但对体系结构的描述、表示、设计和分析以及验证等内容的研究还相对不足，不能处理日益增长的信息量和数据量。因此，面向服务、云的方法就出现来解决这些问题。

⑥ 特定领域的体系结构 DSSA：特定领域的体系结构是将体系结构理论应用到具体领域的过程，如电信软件的体系结构研究、CASE 体系结构、CAD 软件的参考模型、测试环境的体系结构、信息系统的参考体系结构、网络体系结构 DSSA、机场信息系统的体系结构、信号处理 DSSA 等。

⑦ 软件体系结构支持工具：基本上，每种体系结构都有相应的原型支持工具，如 UniCon、Aesop、ArchStudio 等体系结构支持环境，持主动连接件的 Tracer 工具等。

（3）软件体系结构存在的不足：

软件体系结构为解决软件危机而诞生，为构建大规模软件系统提供了基础性构架，但由于体系结构的特殊性，还存在如下问题。

① 缺乏统一的软件体系结构的概念，现在涉及的体系结构概述达数 10 种，这就导致体系结构的研究范畴模糊。

② ADL 繁多，缺乏统一的 ADL 的支持。

③ 软件体系结构研究缺乏统一的理论模型支持。

④ 在体系结构描述方面，尽管出现了多种标准规范或建议标准，但仍很难操作，即可操作性不强。

⑤ 有关软件体系结构性质的研究尚不充分，不能明确给出一个良体系结构的属性或判定标准，没有给出良好的软件体系结构的设计指导原则，因而对于软件开发实践缺乏有力的促进作用。

⑥ 缺乏有效的支持环境。软件体系结构理论研究与环境支持不同步，缺乏有效的体系结构分析、设计、仿真和验证工具支持，导致体系结构应用上的困难。

⑦ 缺乏有效的体系结构复用方案，尽管体系结构是一种高层的系统抽象，并且具有相对的稳定性，但是体系结构又是经验与设计知识的体现。

⑧ 体系结构发现方法研究相对欠缺。由于系统维护、系统演进、环境变化等因素，因此有必要从那些尚不存在体系结构规格说明的系统中逆向提取和恢复系统的体系结构规格说明，即体系结构逆向发现。

⑨ 对新生的面向开源软件的软件开发的构架存在诸多问题，如如何在众多的开源软件中选择满足需求的软件，如何把所选择的开源软件做为软件实体进行构架等。

3．软件体系结构的类型

同时，近几年来，软件体系结构已成为指导软件构架一种基本的方法，也成为软件工程的一个重要的研究领域，许多大型的软件系统基本上采用软件体系结构的思想进行构架。下面就从软件生命周期的软件体系结构、软件需求阶段的软件体系结构、软件设计阶段的软件体系结构、软件实现阶段的软件体系结构、部署阶段的软件体系结构和后开发阶段的软件体系结构进行概述[14]。

（1）软件生命周期的软件体系结构。起初，软件体系结构是解决从需求到实现，到最后

的部署中的中间过程，是以需求来抽象具体的软件模型，并以此构建软件的结构，为软件分析设计提供蓝图。而早期的SA研究主要集中在软件生命周期的设计阶段，关注如何通过SA解决软件系统的前期设计问题，典型的研究点如体系结构描述语言、体系结构风格、体系结构的验证、分析、评估方法等。此后，随着更多不同背景研究者的参与，SA的研究也开始超出软件设计阶段，逐步扩展到整个软件生命周期。如在系统设计前期考虑在需求中引入对体系结构的考虑，在设计后期考虑如何用SA支持系统的实现、组装、部署，以及开发阶段之后的维护、演化与复用等。在软件系统开发的过程中，SA的主要角色包括：支持开发人员之间的交流、直接支持系统开发、支持软件复用等。

（2）软件需求阶段的软件体系结构。需求工程和SA构造是软件生命周期的两个关键活动：需求工程关注如何刻画问题空间，而SA则主要关注如何刻画解空间。所以，需求工程和SA领域中的绝大多数研究工作都相对独立，又相互支持，即没有好的需求刻画，就没有好的SA的构建。然而在需求阶段研究SA，主要有如下两种工作：一是用SA的概念和描述手段在较高抽象层次刻画问题空间的软件需求；二是探讨如何从软件需求规约自动或半自动地变换到SA模型。从而将软件体系结构的概念引入需求规约，这样有助于保证需求规约和系统设计之间的可追踪性和一致性。并从需求模型向软件体系结构模型的转换，主要包括如何根据需求模型构建SA模型，使需求模型与体系结构之间转换，为分析设计阶段打下坚实的基础；以及如何保证模型转换的可追踪性，这就决定了需求模型与体系结构间需要存对应关系和约束关系，即需求模型要为体系结构服务，反过来体系结构要以需求模型为基础。

（3）设计阶段的软件体系结构。设计阶段是SA研究关注的最早和最多的阶段，这一阶段的SA研究主要包括：SA模型的描述、SA模型的设计与分析方法，以及对SA设计经验的总结与复用等。其中：

① SA模型的描述主要包括SA的基本概念，即SA模型由哪些元素组成，这些组成元素之间按照何种原则组织；体系结构描述语言（Architecture Description Language，ADL），在SA基本概念的基础上，选取适当的形式化或半形式化的方法来描述一个特定的体系结构；SA模型的多视图表示，从不同的视角描述特定系统的体系结构，从而得到多个视图，并将这些视图组织起来以描述整体的SA模型。

② SA模型的设计方法是指通过一系列的设计活动，获得满足系统功能性需求，并且符合一定非功能性需求约束的SA模型。

③ 体系结构分析方法是分析SA设计所产生的模型，预测系统的质量属性并界定潜在的风险，从精度上看，体系结构分析方法可以分为两类：一类是基于形式化方法、数学模型和模拟技术，得出量化的分析结果；另一类是基于调查问卷、场景分析、检查表等手段，侧重得出关于SA可维护性、可演化性、可复用性等难以量化的质量属性，也正是体系结构应用的目的。

④ 体系结构设计经验的总结与复用是总结和记录（codify）软件经验是软件工程的重要目标之一，SA的研究也强调对软件设计经验的总结和复用，所采用的主要手段为体系结构风格和模式、领域特定的软件体系结构和软件产品线技术。其中，体系结构风格是描述某一特定应用领域中系统组织方式的惯用模式，作为“可复用的组织模式和习语”，为设计人员的交流提供了公共的术语空间，促进了设计复用与代码复用。领域特定的软件体系结构是领域工程的核心部分；领域工程分析应用领域的共同特征和可变特征，对刻画这些特征的对象和

操作进行选择和抽象，形成领域模型。而软件产品线是指一组具有公共的可控特征（系统需求）集的软件系统，这些特征针对特定的商业行为或者任务。

（4）实现阶段的软件体系结构。最初的 SA 研究往往只关注较高层次的系统设计、描述和性质验证，而对缩小从体系结构层次到系统实现（如代码）层次的鸿沟关注不够。为了有效实现从 SA 设计向实现的转换，现阶段的体系结构研究在以下几个方面进行探索：研究基于 SA 的开发过程支持，如项目组织结构、配置管理等；寻求从 SA 向实现过渡的途径，如将程序设计语言元素引入 SA 阶段、模型映射、构件组装、复用中间件平台等；研究基于 SA 的测试技术等。其中：

① SA 提供了待生成系统的蓝图，根据该蓝图实现系统需要较好的开发组织结构和过程管理技术；同时，SA 还可以用于在开发过程中制定软件开发计划，管理风险和制定相关决策。

② 为了寻求高层 SA 模型和底层实现之间的过渡，研究者们提出了若干方法，其主要思想是尽量封装底层的实现细节，并通过模型转换、精化等手段缩小概念之间的差距。有以下几类典型的方法：

- 在 SA 模型中引入实现阶段的概念，如引入程序设计语言元素等。一般用 ADL 来加以描述，为了促进从设计模型向实现阶段的转化，可以在设计阶段引入实现阶段的概念，即在 ADL 中引入与实现相关的元素。
- 通过模型转换技术，将高层的 SA 模型逐步精化成能够支持实现的模型；即从设计阶段的 SA 模型向代码的转换，是将设计阶段的 SA 模型逐步精化的过程。目前的解决方案或者将高层 SA 模型直接映射成为程序代码，或者经过一系列中间模型的转换，渐进地映射到程序代码。
- 封装底层的实现细节，使之成为较大粒度构件。这时在 SA 设计模型的指导下，选择合适的可复用构件进行组装，可以在较高层次上实现系统，并能够提高系统实现的效率。在构件组装的过程中，SA 设计模型起到了系统蓝图的作用。构件组装是实现软件体系结构的具体实现，但一般在实现时需要构件间的连接子、检测并消除体系结构失配、结合模型转换与构件组装。

③ 基于 SA 的测试技术是作为软件开发的一个重要阶段，测试通过观察在输入一组测试用例的情况下程序的执行行为来动态地验证程序是否正确。可测试性是 SA 的重要属性之一，测试和 SA 之间可以互相借鉴，如体系结构可以用于自动生成测试用例、形成测试计划等；测试能够通过模拟技术评估体系结构模型，评估实现和体系结构规约的相符度。

（5）部署阶段的软件体系结构

随着网络与分布式软件的发展，软件系统的应用都是基于网络而部署的，就使得软件部署逐渐从软件开发过程中独立出来，成为软件生命周期中一个独立的阶段。为了使分布式软件满足一定的质量属性要求，如性能、可靠性等，部署需要考虑多方面的信息，如待部署软件构件的互联性、硬件的拓扑结构、硬件资源占用（如 CPU、内存）等。

（6）后开发阶段的软件体系结构

后开发阶段是指软件部署安装之后的阶段。这一阶段的 SA 研究主要围绕维护、演化、复用等方面来进行，典型的研究方向包括动态软件体系结构、体系结构恢复与重建等。

2.2.1.2　构件的状况及描述方法

构件是软件系统中相对独立的有机组成部分，最初称为模块，是软件体系结构具体表现，

也是软件体系结构实现可复用性的关键。若将各独立的构件实现有效的组装，就需要使构件的关注点分离和层次粒度的重用。因此，就形成了一种基于构件的软件开发方法（Component-Based Software Development，CBSD），但 CBSD 出现不是新的，但其外延基础技术仍在不断发展，从而使构件作为软件系统分解与隔离的一种方法。CBSD 的发展经历了如表 2-8 所示[15]，基于软件体系结构的 CBSD 的可用模型如表 2-9 所示，这些传统经典的软件分析模型也时常指导着基于构件的软件开发，主要表现在对需求进行建模分析、快速获得需求抽象模型、提炼需求的中心要素，以及在分析设计阶段也常采用；但是 CBD（Component-Based Development）往往也离不开这些软件建模方法的来实现，因此将 CBD 也作为一种软件建模方法，但可以将 CBD 作为 CBSD 的一个字集。同时，软件也是一种基于分布对象技术，强调通过可复用构件设计与构造软件系统的软件复用途径。基于构件的软件系统中的构件可以是商业货架产品供应（Commercial-Off-the-Shelf，COTS）构件，也可以是通过其他途径获得的构件（如自行开发），从而将软件开发的重点从程序编写转移到了基于已有构件的组装，以更快地构造系统，减轻用来支持和升级大型系统所需要的维护负担，从而降低软件开发的费用。但随着构件在各类信息化建设解决方案在不同领域的应用，也受到了一些问题的影响：

（1）构件质量的提高和种类的增加；
（2）要求降低系统开发和维护成本的经济压力；
（3）构件集成技术的出现；
（4）软件开发组织内可以用于新系统开发的已有软件制品的数量增加；
（5）受面向服务、面向云的构件的影响。

表 2-8　与 CBCD 相关的软件发展的四个阶段比较

发展阶段	关注点	分布式	思维层次	代表性语言
面向机器	指令、存储	不支持	机器	汇编语言
面向过程	算法、功能	可支持	问题	C、Portan 等
面向对象	抽象、封装	支持	系统	C++、Java、C#等
面向构件	复用、组装	支持	组织	无或 XML
面向服务	发现、选择、组合	全支持	组织、部署	无或 OWL-S、XML
面向云计算	计算、存储、云类型	全支持	组织、结构、部署	无

表 2-9　基于软件体系结构的 CBSD 的可用模型比较

特征 / 类型	类型重用	组件重用	互操作性	可升级性	复杂性	时间周期	可靠性	质量
瀑布模型	不能实现	不能实现	没有	不可以	高度复杂	长周期	低	低
原型模型	不存在	没有工件	未规定	不可以	高度复杂	不独立用	低	无
RAD 模型	不能实现	不存在	不支持	不可以	高度复杂	快速开发	弱	低
演化模型	可实现	难实现	不互通	可能	高度复杂	长周期	弱	中
OO 模型	可实现	没特性	允许	可以	一般	长周期	高	中
RUP 模型	可实现	不包含	高规定	可以	一般	大量时间	中	中
敏捷模型	可实现	无属性	限制	可以	一般	快速开发	低	低
CBD 模型	高度支持	核心功能	高度支持	可以	不复杂	快速开发	高	高
面向服务模型	高度支持	全支持	高度支持	完全可以	不复杂	可快速开发	高	高
面向云模型	高度支持	全支持	高度支持	完全可以	不复杂	可快速开发	高	高

在表 2-9 中：

（1）瀑布模型：瀑布模型核心思想是按工序将问题化简，将功能的实现与设计分开，便于分工协作，即采用瀑布模型用结构化的分析与设计方法将逻辑实现与物理实现分开，最终使整个软件开发流程形成一个“瀑布型”的结构。

（2）原型模型：先借用已有系统作为原型模型，通过软件原型不断改进，使得最后的产品就是用户所需要的软件系统。

（3）RAD 模型：快速应用开发（RAD）是一个线性顺序的软件开发模型，强调极短的开发周期。主要关注软件的业务建模、数据建模、处理建模、应用集成和测试反复。

（4）演化模型：演化模型是一种全局的软件（或产品）生存周期模型。属于迭代开发风范，如需求→设计→实现→测试→集成→反馈的流程，直到满足需求为止。

（5）OO 模型：是当前计算机界关心的重点，也是目前软件界使用最广泛的一种方法，并且已超越了程序设计和软件开发，扩展到很宽的范围。它是一种把面向对象的思想应用于软件开发过程中，主要包括对象、类、消息，其特征主要包括封装性、继承性、多态性等。

（6）RUP 模型：统一软件开发过程模型（Rational Unified Process，RUP）是一个面向对象且基于网络的程序开发方法论，它可以为所有方面和层次的程序开发提供指导方针、模版以及事例支持。

（7）敏捷模型：是一种态度，而不是一个说明性的过程，即敏捷建模者们坚持的价值观、敏捷建模者们相信的原则、敏捷建模者们应用的实践组成的集合。

（8）CBD（Component-Based Development）模型：基于组件的开发是一个以组件为基础的软件开发模式，一旦体系结构被建立，它必须用组件去充实，这些组件或者可从复用库中获得，或者根据专门需要而开发。一般包括有 EJB（Enterprise Java Bean）、.NET、COM（Component Object Model）/COM+、CORBA（Common Object Request Breaker Architecture）等。

（9）面向服务模型：是近来兴起的一种新型软件建模方法，它主要将软件系统中的功能以服务的形式发布给用户，并在进行需求建模时，用服务的思想来进行分析。

（10）面向云模型：它是一种可伸缩、可存储、可计算、可使用的网络化模型，即是将大量用网络连接的计算资源统一管理和调度，构成一个计算资源池向用户按需服务。

CBSD 整个过程从需求开始，由研发团队使用传统的需求获取技术建立系统的需求规约，并在完成体系结构设计后，并不立即开始详细设计，而是确定哪些部分可由构件组装而成，即选择什么样的构件库或构件系统来支持所需求的软件体系结构。但此时研发会面临如下决策问题：即“是否存在满足需求的构件”，“是否存在满足某种需求的内部开发的可复用构件”，“这些可用构件的接口与体系结构的设计是否匹配”等。对于那些无法通过已有构件满足的需求，就只能采用传统的或面向对象的软件工程方法开发新构件，以及采用面向服务的模式重新构造新的构件，这样就使构件的可复用性大大提高，使关注点分离更有效。而对于那些满足需求的可用构件，开发人员通常需要进行如下活动：

（1）构件选择与鉴定：需要根据需求选择同类型可用的构件，然后对所选择的构件进行鉴定。构件鉴定一般分为发现和评估两个阶段，其中发现阶段需要确定构件的构件接口的功能性（构件能够提供什么服务）及其附加属性（如是否遵循某种标准）、构件的质量属性（如可靠性、可用性、健壮性）等属性。但往往构件发现难度较大，因为构件的属性往往难以获取，难以采用一种量化的评估方法加以准确的判别。而评估阶段根据构件属性以及新系统的

需求判断构件是否可在系统中复用及关注点是否实现有效的分离。评估方法常常涉及分析构件文档、与构件已有用户交流经验、甚至开发系统原型。构件鉴定有时还需要考虑非技术因素，如构件提供商的市场占有率、构件开发商的过程成熟度等级等。同时，构件的选择与鉴定是同步进行，这样才能有效保证 CBCD 的开发进度。

（2）构件适配：如果被复用的构件不符合目标系统的软件体系结构就可能导致该构件无法正常工作，甚至影响整个系统的运行，也不能满足 CBCD 的要求，这种情形称为失配。调整构件使之满足体系结构要求的行为就是构件适配。其构件适配可通过白盒、灰盒或黑盒的方式对构件进行修改或配置，从而达到目标系统的体系结构和满足具体的需求。白盒方式允许直接修改构件源代码；灰盒方式不允许直接修改构件源代码，但提供了可修改构件行为的扩展语言或编程接口；黑盒方式是指调整那些只有可执行代码且没有任何扩展机制的构件。如果构件无法适配，就不得不寻找其他适合的构件，并重新时行选择和鉴定。

（3）构件组装：构件组装技术是基于构件的软件开发的核心技术，构件必须经过组装才能形成软件应用系统，才能实现软件的价值。但构件必须通过某些良好定义的构件系统和构件库才能组装成目标系统。软件体系风格决定了构件之间连接或协调的机制，是构件组装成功与否的关键因素之一，典型的体系风格包括黑盒、白盒、消息总线、对象请求代理等，它们间的比较如表 2-10 所示。

表 2-10　构件组装技术比较

特征 种类	编织能力	体系结构配置	粘连码	标准化
白盒方法	支持	不支持	不支持	支持
黑盒方法	不支持	不支持	不支持	支持
框架方法	支持	支持	不支持	支持
连接器方法	支持	支持	支持	支持
胶粘码方法	支持	支持	支持	支持
总线方法	不支持	不支持	不支持	支持

（4）构件更新：基于构件的系统演化或重组往往表现为构件的替换或增加，其关键在于如何充分测试新构件以保证其正确工作且不对其他构件的运行产生负面影响，对于由构件组装而成的系统，其更新的工作往往由提供构件的第三方完成。同时，构件能动态更新，也是对 CBCD 的软件系统进行升级，从而提供应用软件的扩展性和可复用性。

下面就从以上三个方面进行一步描述构件的特征，主要包括构件形式化定义、构件组装模式和应用案例。

1．构件形式化定义

根据构件的性质给出以下构件形式化定义：

定义 2-1　如果刻面采用一棵有向树来描述[17]，则刻面分类信息 S 可以是一个四元组，记为 $S=(m,n,v,sE)$，其中 m 是一组刻面分类集合的名称，n 为 m 的刻面名称，v 为刻面的术语名称，sE 表示语义。

定义 2-2　构件用一个四元组 $C=(cn, PI, RI, S)$，其中 cn 为构件的名称，PI 为构件的提供的接口集，RI 为构件的请求接口集，每个接口是由一组服务构成，即这些接口是由 Web 服务

接口实现构件组装的，同时，这里的服务既可以是被调用的方法，也可以是异步的消息。每个构件可以不包含请求接口，但是至少要包含一个提供接口。

定义 2-3 构件接口集元组 CI $(\{C(RI,RI)\}, P, B, FS, EP)$描述，其中，$\{C(RI,RI)\}$为构件的请求与提供组成的 Web 服务接口集，$P$ 为构件的行为特征，B 为构件的功能实体，FS 为构件的功能描述，EP 为构件的功能实体点，且 $B=\{EP_1, EP_2,\ldots,EP_m\}$，

定义 2-4 构件扩展可由一个二元组描述：$CE=(C, CI)$。

定义 2-5 构件刻面机制用一个四元组描述，记为 $FC=(m, CE, O, IF)$，其中 O 是刻面操作符，IF 是刻面检索条件，m 是刻面分类集合，见定义 2-1。

定义 2-6 构件结构 CA 可以是一个五元组，记为 $CA=(ID,Type,Tier,Dom,RT)$，其中，ID 是构件结构标识符；$Type$ 表示构件模型，如设计模板、代码等；$Tier$ 表示层次，如表示层、业务层、数据层等；Dom 表示域，区分刻面属于何种域；RT 表示反馈策略。

定义 2-7 设 CL 为构件库，$CE_1=(C_1, CI_1), CE_2=(C_2, CI_2)$为 CL 中的两个构件，若对任意接口 $ci_1\in CI_1$ 为构件接口提供一个请求接口，则在 CL 中至少存在一个构件 $ci_2\in CI_2$ 为一个提供接口，若满足 $ci_1\rightarrow ci_2$，则称 CL 构件库可用且完备。

定义 2-8 构件组装可以用 $CC=(CL_1\{CE_{1i},CE_{2i},\ldots,CE_{ni}\}, CL_2\{CE_{1j},CE_{2j},\ldots,CE_{nj}\}, CA, FC)$。

定义 2-9 给定概念 $C_{p1},C_{p2},C_{q1},C_{q2}$，存在关系 r_1,r_2，使得两个相关关系$\chi_1=< C_{p1}, r_1, C_{q1}>$，$\chi_2=< C_{p2}, r_2, C_{q2}>$，使得满足定义 2-1、定义 2-5 中的刻面操作和检索中的语义识别，若存在 $\chi_1:CE_1\rightarrow CL_1$，$\chi_2:CE_2\rightarrow CL_2$，使得 $sE.\chi_1 \rightarrow sE.\chi_2$，就表明构件间建立了语义关联，记为 $CE_1\infty CE_2$。

定义 2-10 设 SC 为一面向构件的应用系统，则 SC 可描述为 $SC=(CL,CC,\{ CE_1\infty CE_2\})$。

2．构件组装模式

构件组装是基于构件的软件开发的核心技术，也是 CBSE（Component-Based Software Engineering）开发的一个核心过程。就使得构件必须经过构件组装才能生成所需的软件应用系统，才能实现构件的价值。而构件通常是根据构件刻面分类模式、检索方式、构件寻址、匹配模式和动态反馈机制等实现组装，这是因为构件库中的每个构件的功能区别描述一种特征，且这种特征是每个不同构件所特有的属性，也就是一个构件对应一个特征，然后根据每个构件的特征描述一种接口，并将刻面置入这种定位接口中完成构件组装。同时，而一个具体的构件主要由构件接口和描述规约两部分组成。一个接口提供一种服务，完成某种逻辑行为，以及实现各构件接口连接其他构件、特征和行为的识别；而构件接口和行为是一种构件本身和具体行为的描述。这些构件本身就是该构件具体的业务逻辑，而行为是通过有效的语法和语义的识别，以达到各构件准确的定位到自己所需的构件。但从当前构件研究来看，构件组装是构件技术研究的重点，这是因为构件组装从构件生产服务，对构件技术的发展形成了强大的约束。并且构件组装往往是通过构件所提供的 Web 服务接口进行关系，最终获得构件的功能。

当然，构件组装的前提首先得进行构件选择和鉴定，再在构件库中进行适配到请求的功能，也说明了构件本身的编程难度大不，而是如何将构件组装成应用软件系统却受到构件模型、体系结构、构件关注点、构件粒度和运行环境等影响，因此，在进行构件组装时，不但要注重构件的外部因素，还得考虑内部细节，即包括基于构件的软件开发的体系结构和组装结构模式，还要根据不同的外部因素考虑构件组装的组装方式，以及内部的连接子、语义关

联等。下面主要介绍构件的组装结构模式，包括敏捷方法、正式方法和需求定制方法。图 2-19 是一种构件组装体系结构。

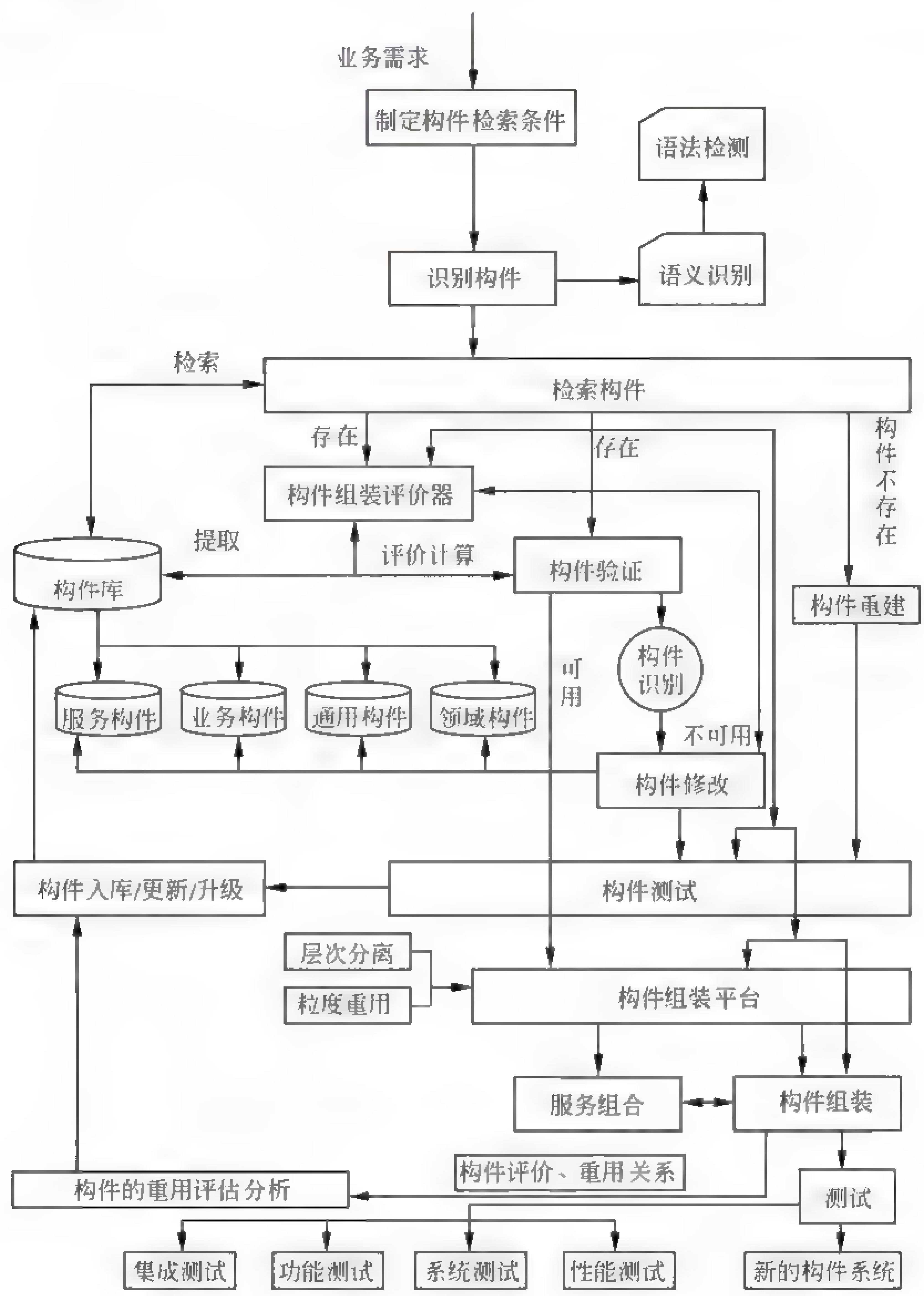

图 2-19　构件组装体系结构

构件组装模式可能由项目的组织或本质决定，也可以根据需求来决定。使得在专门命令和控制严格且灵活的技术之间求得平衡成为完成应用程序开发项目的一个关键组成部分。但有用且有效的开发方法应该包括从松散托管到完全指定的各种技术。所使用的技术类型和所应用的流程精确量由应用程序设计、技能集合、技术成熟度、规模、复杂性和重要性中涉及的风险决定。这些技术的一端是开发团队，将与涉众紧密合作，创建满足一组已经标识并进行了优先排序功能的应用程序解决方案。

（1）面向敏捷方法的构件组装模式。敏捷开发方法是轻量级流程，追求尽可能减少标识

需求与工作代码交付之间的时间延迟。而抽象需求捕获是为所需功能的通用声明或用户案例（以叙述的方式描述如何使用系统实现目标）服务，使得需求正式化是实现的测试用例。敏捷开发的目标就是通过测试用例来实现，其目的是尽可能少的流程的开销，从而降低业务逻辑交互所需功能。

面向敏捷开发流程每次处理少量的用户案例或需求，并让敏捷开发中的计划调度基本上就是一个时间框方法，使其完成的交互和需求能在规定的时间范围内。而敏捷开发中的每次迭代空间应该满足交互需求，以达到至少能交互应用程序的一个重要功能方面的内容，但这时迭代空间不要太大而分散敏捷开发团队的注意力。

这种方法非常适用构件组装过程中，且需求较为明确，用户功能易于获得的软件组装领域。图 2-20 所示是面向敏捷方法的构件组装模式。

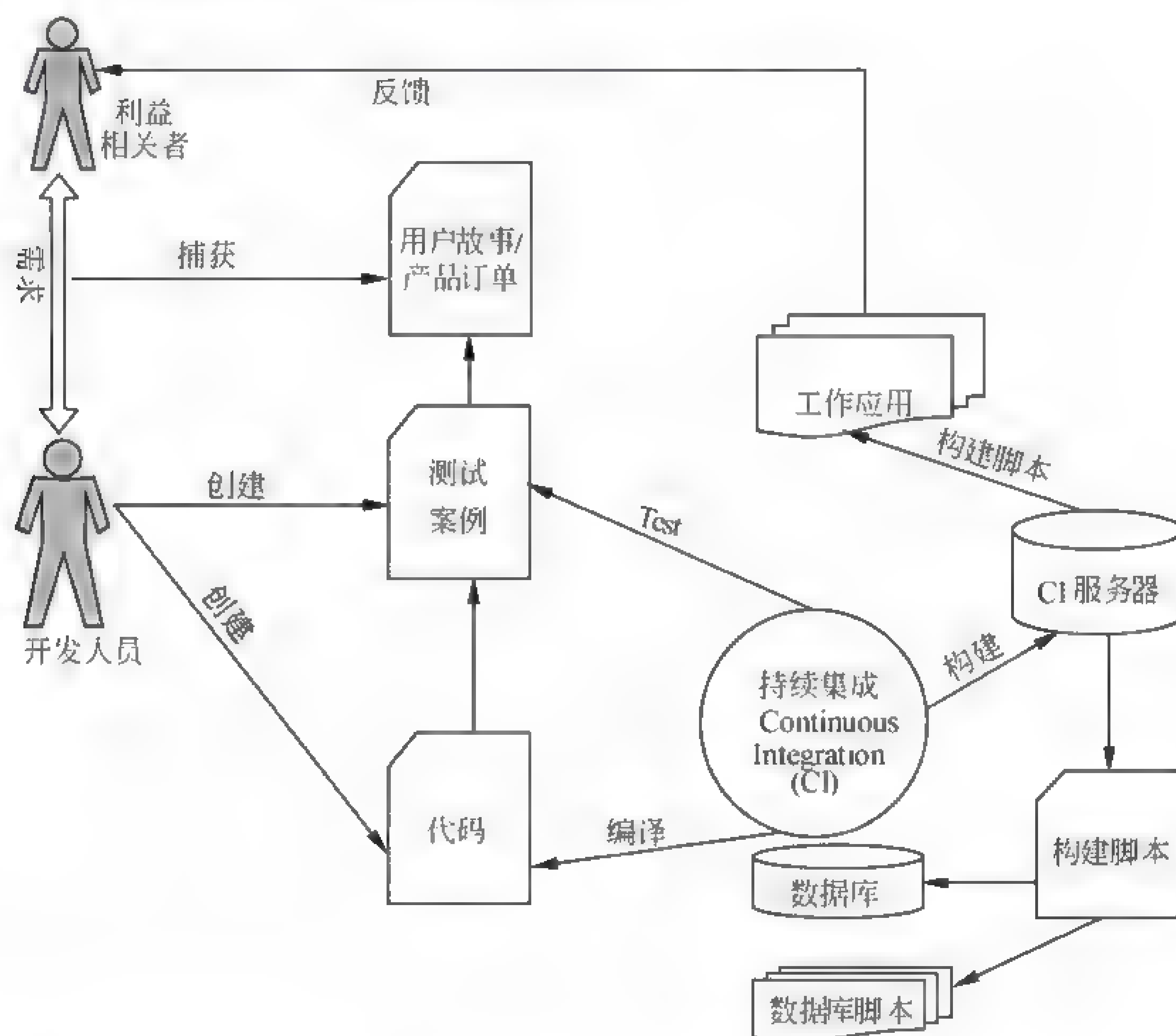

图 2-20　面向敏捷方法的构件组装模式

（2）面向正式的方法的构件组装模式。面向正式的方法是严格根据软件工程而设计的（见图 2-21），它将软件研发工作划分为具体的规程（分析、设计、编码、测试等），使得涉众提供的信息以及对涉众的反馈更为程序化和正式化，并且直接性和连续性更低一些。当对构件的需求得到了很好地理解，且很少或没有可重用设计与实现资产可用时，就可以使用这种类型的方法来实现构件组装。但这时需要更注重项目的管理，这是因为采用正式方法进行构件组装时的前提是对需求很好理解且几乎没有可重用设计和实现，项目管理可以促使面向正式的构件模式更有效。即：

根据组织的不同，架构师可能不会对应用程序开发项目的日常管理负有最终责任。不过，应该清楚地了解与项目预算和计划管理相关的问题和需求，并且当在体系结构的帮助下进行

准确的计划和调度时，应该能够分析组件固有的复杂性和风险影响因素。还可以为特定的组件指定特定的技能或技能级别，从而帮助准确地进行资源调度。

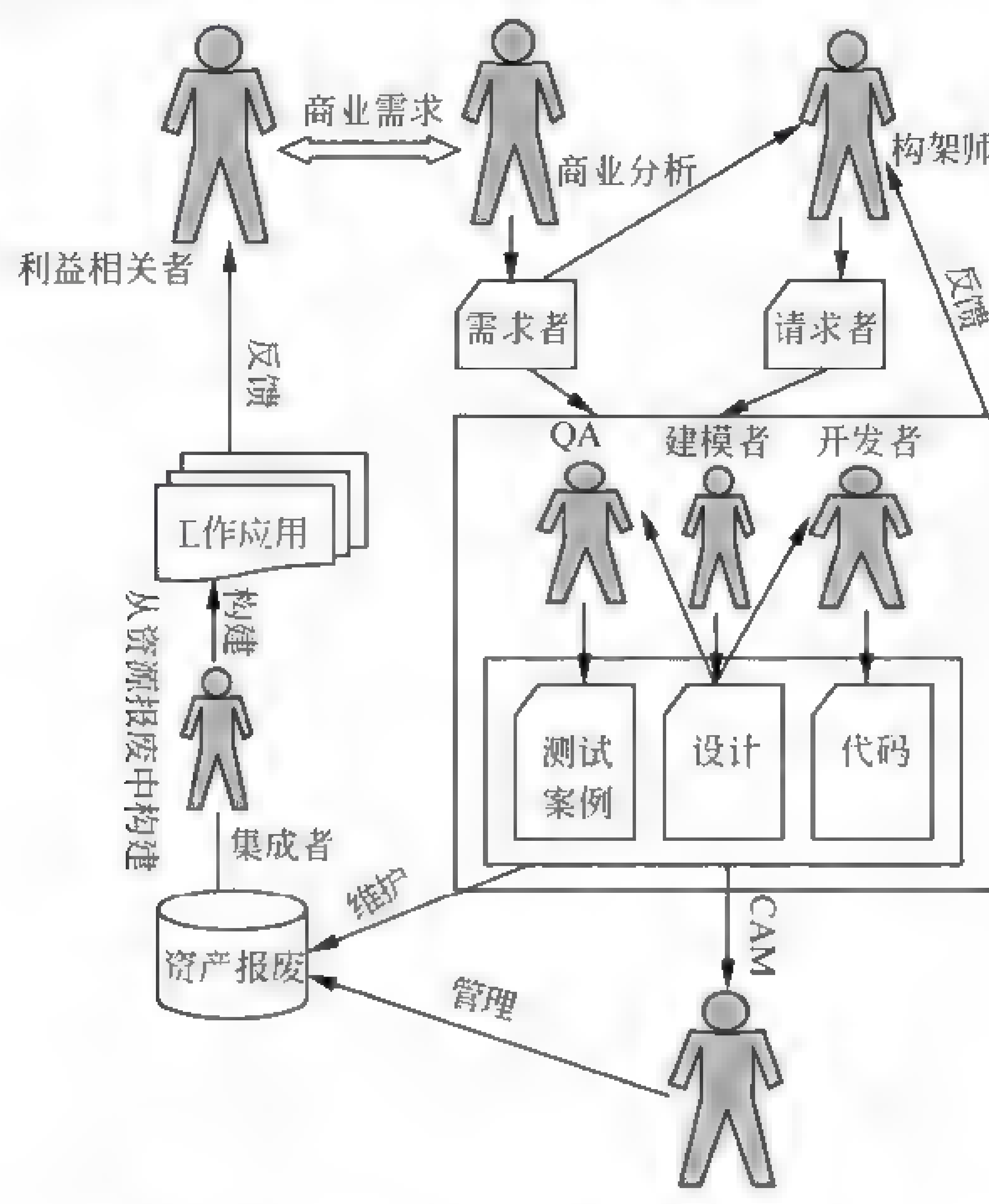

图 2-21 面向正式的方法的构件组装模式

（3）面向需求定制的构件组装模式。当流程精确度要求的另一端是产品线体系结构时，按需的应用程序设计、流程构造、配置和部署都应均为预先开发的，这与其他类型的“生产线工程”有些类似。这时，若材料、部件和组装流程都已标准化，这就为按需求定制的构件组装模式提供的基础结构。

但在任意规模的项目中，会很明显地发现，无论如何在多方面进行精心的定制，都不会有同时适合应用程序的所有方面的单个方法；因此，就导致体系结构的有些组件具有模糊或未知的需求，这样组织或开发团队会随着时间不断发展进行可重用组件设计。并在进行此工作的过程中，有些应用程序类型的开发方法会从试验性的创造流程过渡到正式的重用和变体。因此，了解各种技术的细节非常有价值，但还要确保了解项目的其优缺点。

另外，当选择应用程序的开发技术，以及在该技术中进行开发工作时；体系结构本身充当将应用程序划分为反映关注分离的结构的主要机制。这个分离允许架构师以独立于其他组件的方式分析每个组件的特征。若要应用的方法只是其中的一个特征，应该由体系结构分析进行标识和提供支持。图 2-22 所示是面向需求定制的构件组装模式。

3. 应用案例

目前构件应用取到了丰硕的成果，包括国外的 OMG 的 CORBA、Microsoft 的 COM、SUN 的 JavaBeans/EJB，国内的北京大学的青鸟构件库，上海普元构件库等，并且提出了各自的构件组装方法，如 OMG 的 CMM 方法，青鸟构件库的 ABC 方法等。各自都开发了不同的领域性构件库，如上海普元的电信行业构件库等。而构件按层次可分为服务构件和业务构件两大

类，如图 2-23 所示，但从构件组装角度目前仍处于手工或半自动化组装方式。

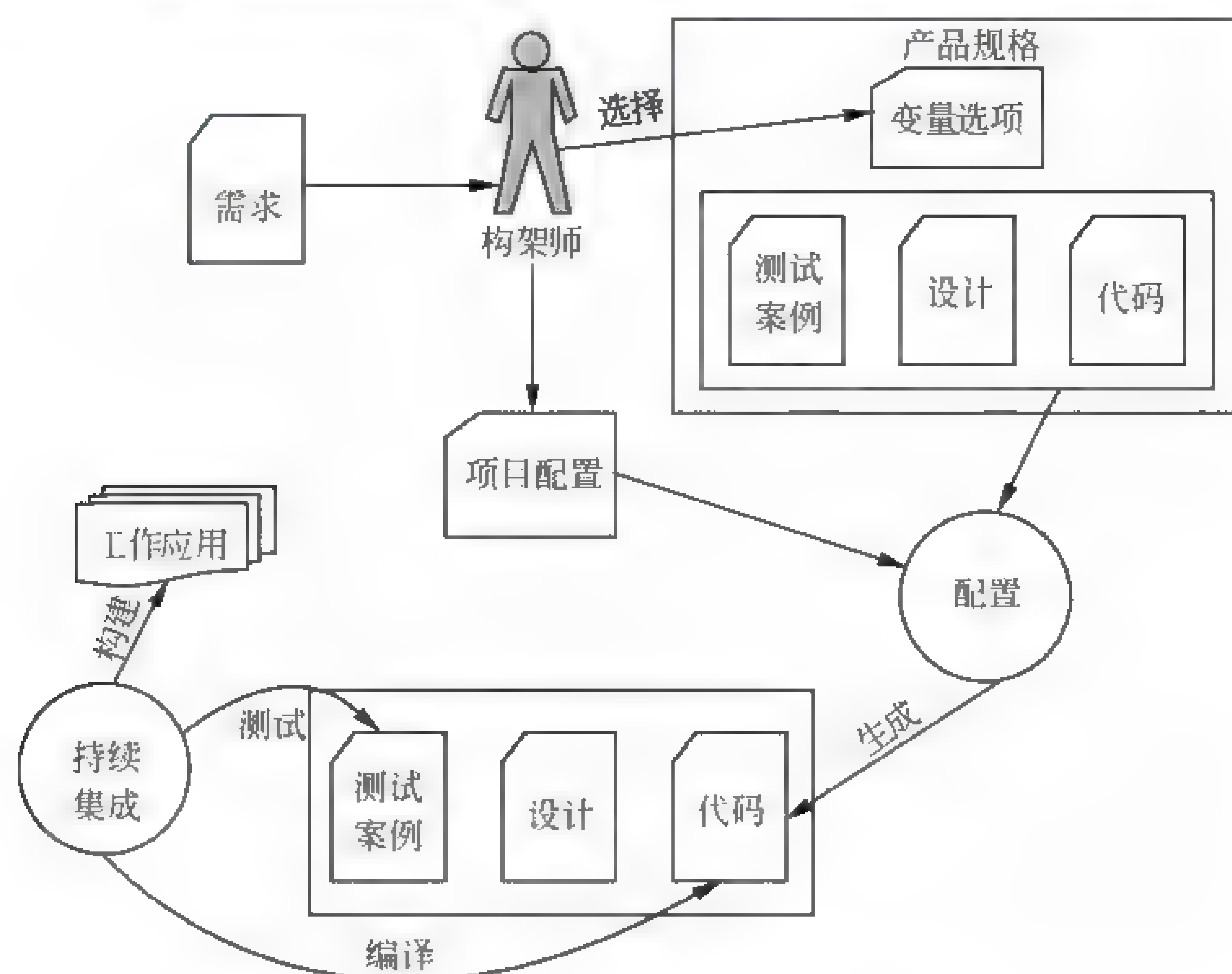

图 2-22　面向需求定制的构件组装模式

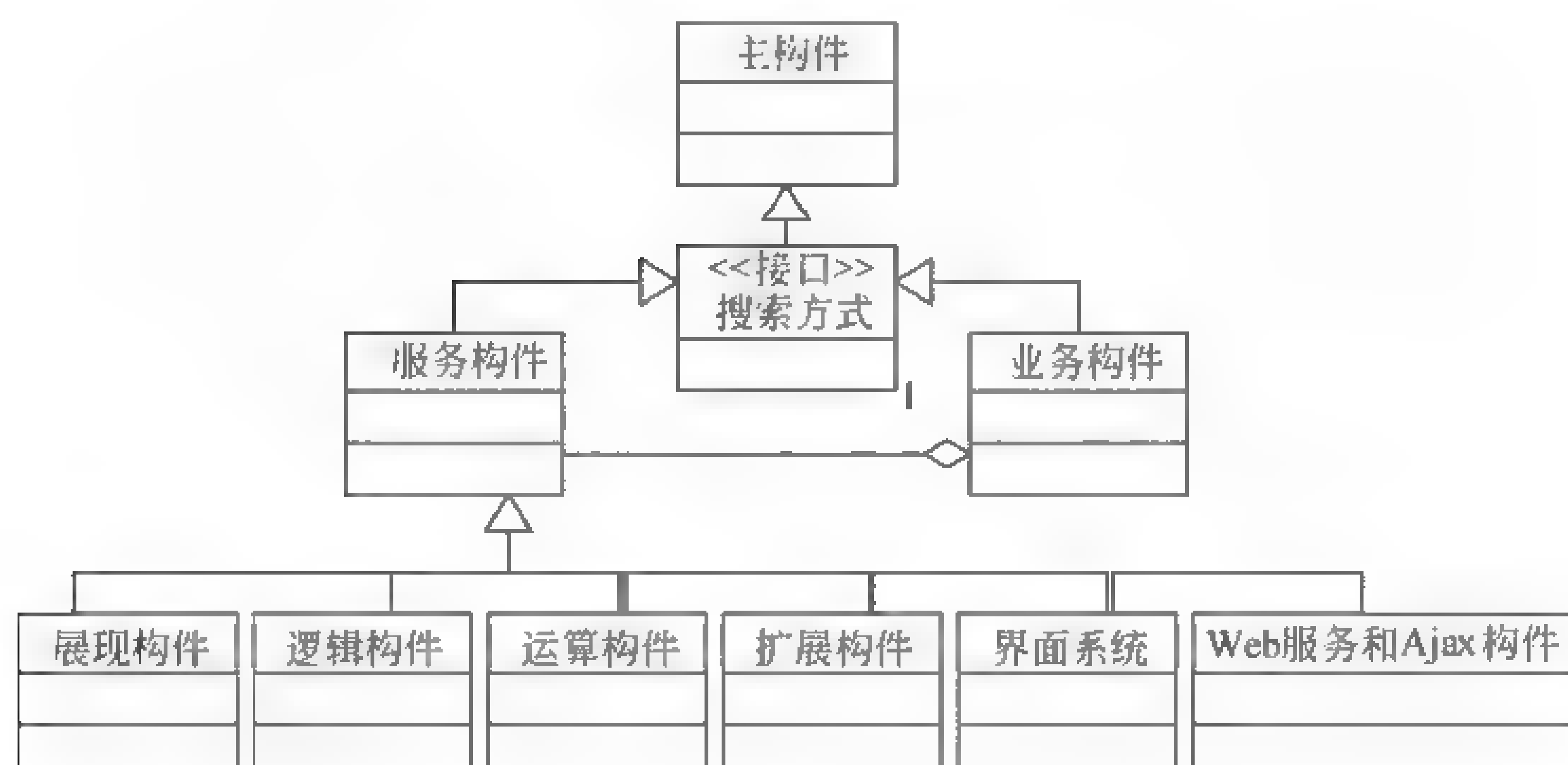

图 2-23　构件类型

但构件间的互通、互操作、跨度应用、可自动集成组装等方面还显得很不够。同时，新型网构软件（internetware）的语义化、组装智能化等领域还面临巨大的挑战。为了从不同程序上解决这些问题，基于 XML 相关的描述提供了解决语法的方面问题，为描述逻辑关系更为方便，求解问题更加严密。而且 RDF(S)实现了基本资源的描述；WORF（Web Services Object Runtime Framework）、WSRF（Web Services Resource Framework）和 WS-*系列等为分布式运行环境、资源描述和 Web 服务技术标准进行了规约；Ontology 对资源间、概念间的关系实现了形式化语义描述。因此，在美国军方主持下的 ALOAF 构件库实现了资源共享和无缝互操作，并提交了 ALOAF（开放体系结构的构件库框架）报告，在此基础上，RIG 开发了 UDM（Unified Dimensional Model）/BIDM（Basic Interoperability Data Model）软件复用共享构件库，

以及 NATO（North Atlantic Treaty Organization）制定了一组关于软件复用的标准。国内的北大青鸟构件库项目成员提出许多具有创新和开创性的方法来实现构件复用、共享、互通、组装等。

下面从 CORBA、北大青鸟构件、上海普元构件这三个当前应用广泛的常用构件进行概述，并用一个校园 MIS 构件库为例进行分析。

（1）CORBA 简介。CORBA（Common Object Request Broker Architecture）构件是基于 CCM（CORBA Component Model）的应用系统中构件块，由对象管理组织（OMG）设立并进行控制，CORBA 定义了一系列 API，通信协议，和对象/服务信息模型来使得异质应用程序能够互相操作，这些应用程序用不同的编程语言编写，运行在不同的平台上[①②]。因此 CORBA 为定义明确的对象提供了平台和位置的透明性，这些对象是分布式计算平台的基础。同时，CORBA 把用其他语言开发的程序代码和关于该程序代码能力和如何调用该程序代码的信息包到一个开发包(package)中，开发包中的对象则可以在网络上被其他程序(或 CORBA 对象）调用。并使用一种接口定义语言（IDL）用于刻画对象将体现出来的接口，其 IDL 语法结构见[③]。CORBA 又规定了从 IDL 到特定编程语言，如 C++或 Java，并通过 IDL 实现的映射，这个映射精确的描述了 CORBA 数据类型是如何被客户端和服务器端实现的，如图 2-24 所示。

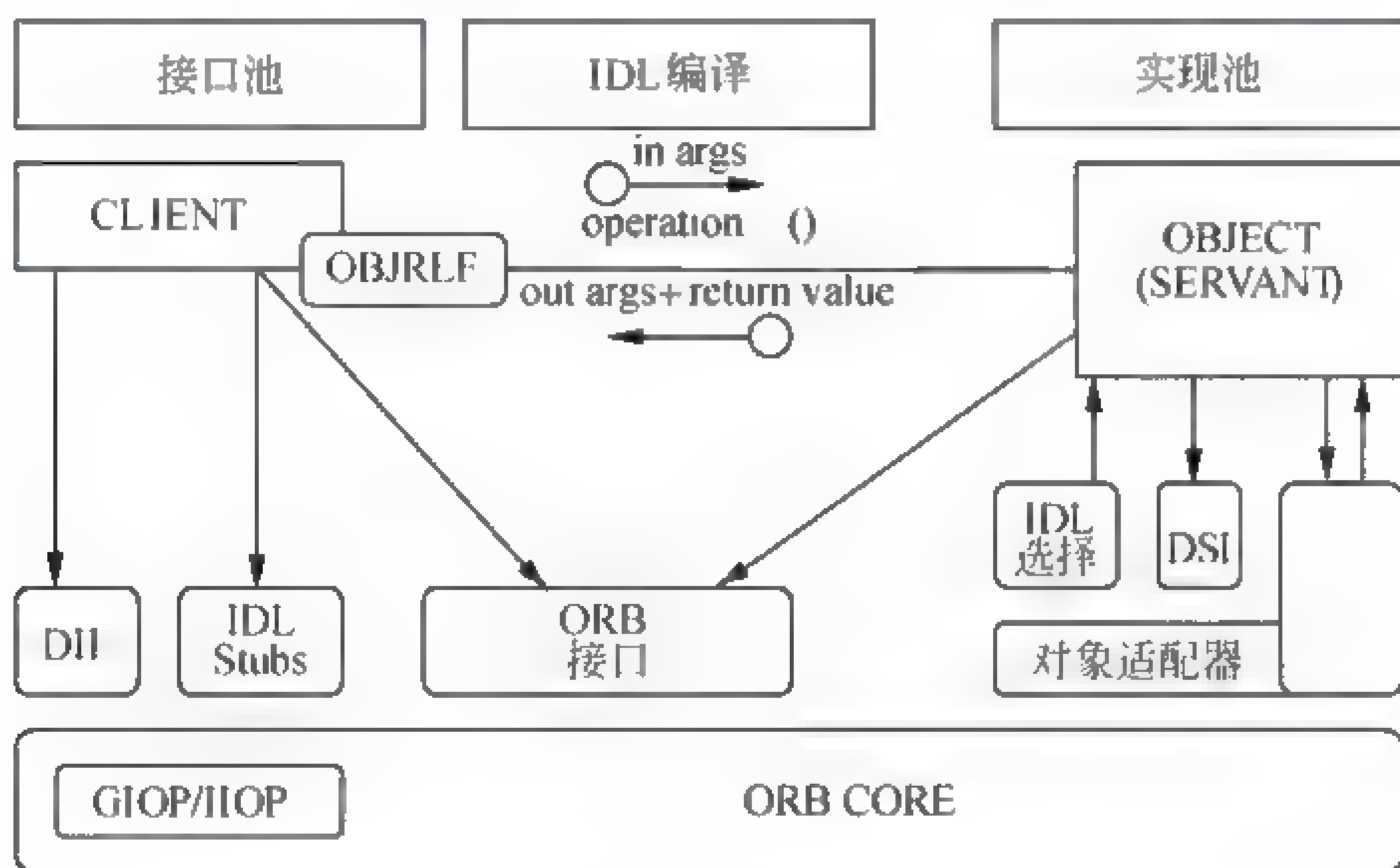

图 2-24 CORBA 体系结构
（来源：http://www.omg.org/spec/CORBA/3.1/Interfaces/PDF/）

因此，CORBA 可以总结为下以特征：

- CORBA 定义了一种面向对象的软件构件构造方法，使不同的应用可以共享由此构造出来的软件构件，并通过 IDL 进行各构件进行组装。
- 每个对象都将其内部操作细节封装起来，同时又向外界提供了精确定义的接口，从而降低了应用系统的复杂性，也降低了软件开发费用，从而提供软件可复用性。
- CORBA 的平台无关性实现了对象的跨平台引用，开发人员可以在更大的范围内选择最实用的对象加入到自己的应用系统之中。

① http://www.longen.com/A-D/detaila-d/CORBA.htm
② http://www.corba.org
③ http://www.omg.org/spec/CORBA/3.1/Interfaces/PDF

- CORBA 的语言无关性使开发人员可以在更大的范围内相互利用别人的编程技能和成果，使其是实现软件复用的实用化工具。
- CORBA 易于理解，具有完整的语义特征；易于扩充和修改，具有较高的通用性和适应性；易于构造组装，具有规范的外部接口等优势。

而 CORBA 主要由对象连接、IDL、动态调用接口、接口公用库等四部分来实现构件组装。目前是 2008 年发布的 CORBA3.1 版，这个版本进一步完善和扩展了 Java 和 Internet 集成、服务质量（QoS）控制和 CORBA 组件体系等内容，这样不仅大大增强了服务器软件的可复用性，而且为 CORBA 应用的动态配置提供了更大的灵活性。

（2）北大青鸟构件简介[①]。青鸟工程是国家重点支持的科技攻关课题。青鸟工程面向我国软件产业基础建设的需求，以实用的软件工程技术为依托，研究开发具有自主版权的软件工程环境，为软件产业提供基础设施、软件工具、平台和环境，建立了工业化生产的基本手段，促进我国软件开发由手工作坊式转向用计算机辅助开发，以提高软件开发效率，改善软件产品质量。

并且青鸟团队在已有的基础上还研究软件的工业化生产技术，开发软件工业化生产系统——青鸟软件生产线系统，即基于构件构架模式的软件开发技术及系统，为软件开发提供整体解决方案，推行软件工业化生产模式，促进软件产业规模的形成。

与此同时，青鸟团队还开发了基于异构平台、具有多信息源接口的应用系统集成（组装）环境青鸟 III 型（JB3）系统，如图 2-25 所示。青鸟 III 型系统研制的目标是针对软件工业化生产的需求，完善并初步实现青鸟软件生产线的思想，制定软件工业化生产标准和规范，研究基于“构件—构架”模式的软件工业化生产技术，研制支持面向对象技术，支持软件复用的，基于异构平台、具有多信息源接口的应用系统集成（组装）环境。

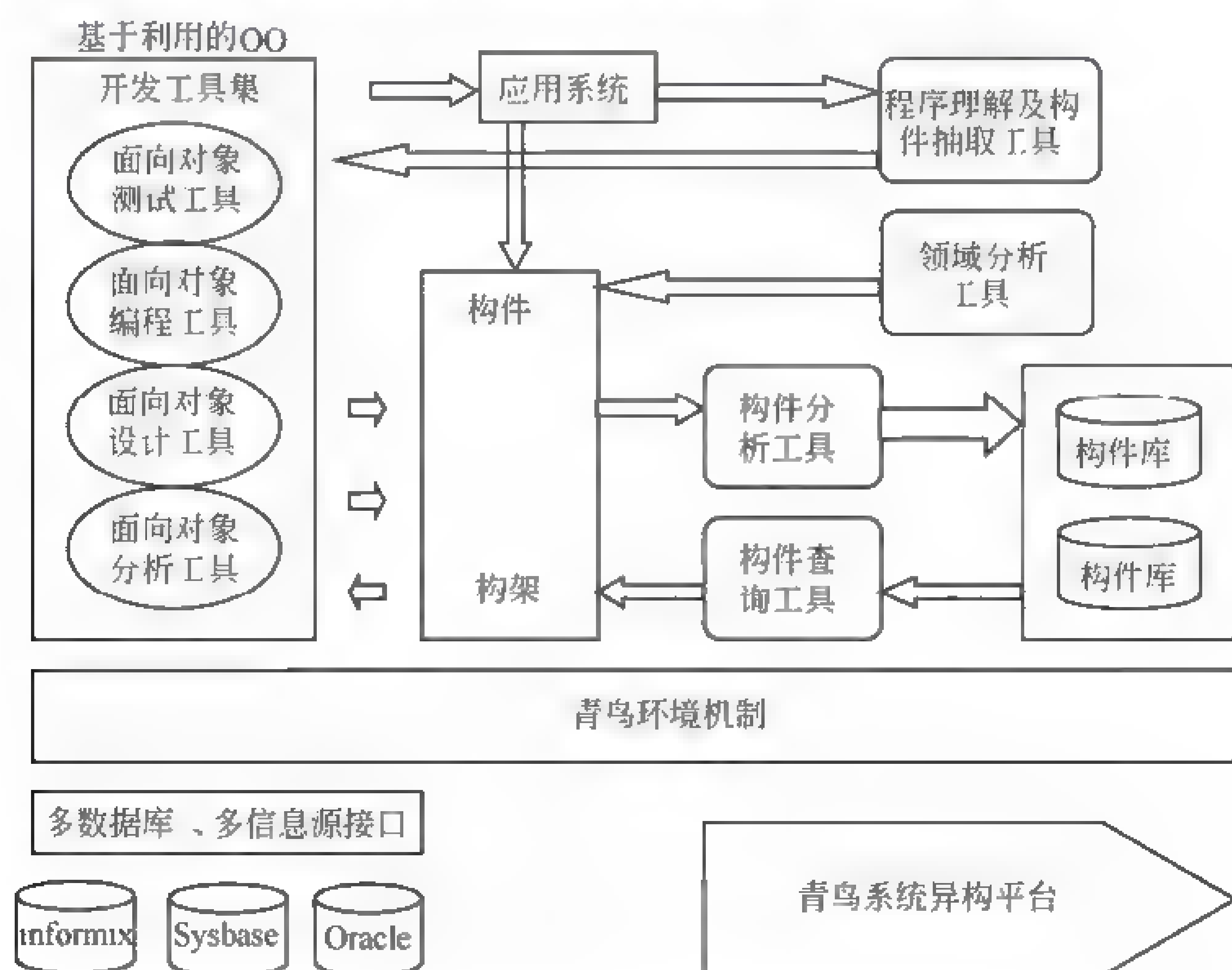

图 2-25　青鸟 III 型系统的体系结构
（来源：http://ftp.sei.pku.edu.cn/jadebird/jb.html?reload coolmenus）

① http://ftp.sei.pku.edu.cn/jadebird/jb.html?reload coolmenus

（3）上海普元构件简介。[①]普元软件（Primeton）是全球领先的 SOA 中间件和构件基础软件厂商，它起初了提供了诸如电信领域的构件库。目前已是 SOA 国际标准 SCA/SDO 的主要参与制定者，以及电子商务标准的主要制定者 OASIS 的核心奠基成员。使得普元产品已经在电信、金融、政府、国防、能源、物流、制造等多个行业和领域的一千余个关键应用上得到验证，并且在国外也拓展大量业务。

（4）校园 MIS 构件库。把校园分散的资源进行融合确定关注点分离和粒度重用来实现基础构件库的设计。其用不变量和变量来确定其分离关注点以及粒度重用，用 Web Service 标准技术完成整个构件库的技术融合、用 Ajax（Asynchronous JavaScript and XML）完成异步实时性包装。在校园中，教师和学生的属性在某一个时期内是处于静态的，这个时期称之为不变量，而他们的活动是使校园活动起来，这些量我们称之为变量，校园中与外界联系的量称之为不确定量。同时在某一时期，教师的职能也是确定的，当然学生的职能在某一个时期也是确定的。因此他们完成的活动事务也确定的，即是不变量。从而教师和学生进行业务活动时，根据自己的特征就可以检索完成关注点分离和粒度重用，如图 2-26 所示。

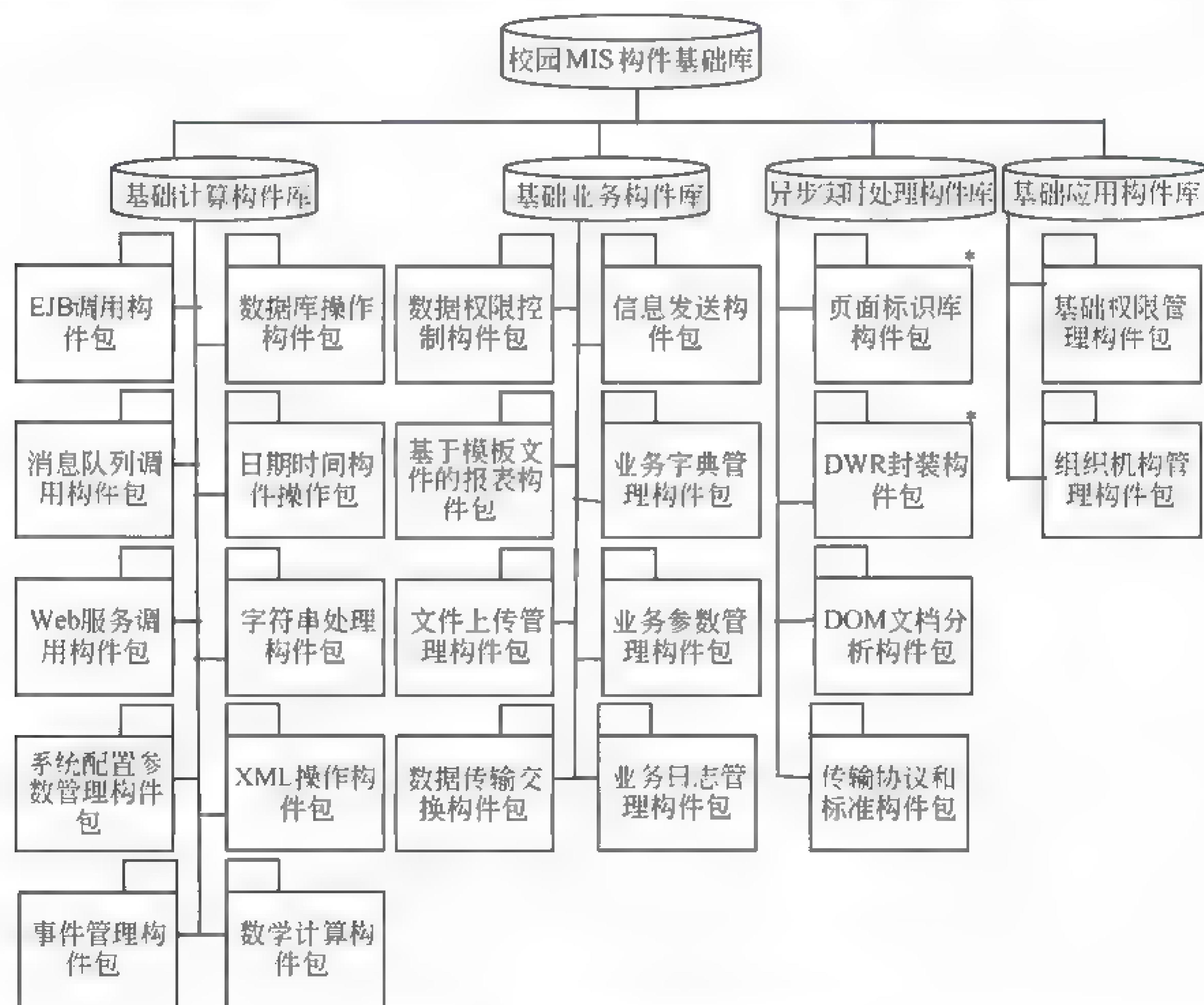

图 2-26　校园 MIS 基础构件

在图 2-26 中：

① 基础计算构件库：在软件体系结构中处于最底层相关的一组连接处理库和实现关注点分离的接口库。包括数据库操作构件包，日期时间构件操作包，字符串处理构件包，XML 操作构件包，数学计算构件包。事件连续处理构件包，EJB 之间调用的构件包，基于 JMS 的

① http://www.primeton.com/index.php

消息队列调用构件包，Web 服务中 UDDI 的注册、查找、描述的构件包和连接不同系统的 XML 配置构件包，这些构件的封装都是基于第二节中的技术路线。

② 基础业务构件库：是一组实现基础计算库构件中的构件包基本运行方式，完成最底层中的各包通信和应用中业务功能相关的操作。包括信息发送、接收构件包，业务检索交换数据的字典管理构件包，业务运行流程参数管理构件包，业务动作的日志管理构件包，基于角色访问控制的、层次的、静态与动态约束的权限管理构件包，基于模板文件报表打印和操作的构件包和文件上传管理构件包。

③ 异步实时处理构件库：包装 Ajax 技术和使用方法的构件包，包括基于 DOM 的页面标识库构件包，Java 类转换为 JavaScript 的 DWR 开源件的构件包，DOM 文档分析与解析的方法构件包以及传输协议、WS-*规范约束和相关配置的构件包。

⑤ 基础应用构件库：实现业务系统安全有效完成一项作业的构件库，包括基于任务的、基于角色的、任务－角色等一组权限管理构件包，且随着不同的访问控制出现而进行升级该库；在校园内，教师这个角色是有组织结构的，因此有组织管理构件包。

图 2-27 所示是校园 MIS 业务构件。主要包括人员资料构件库，教务管理构件库，财务管理构件库，人事管理构件库，图书管理构件库，后勤管理构件库，科技管理构件库，工作协同构件库，业务管理构件库，数据分析构件库等。

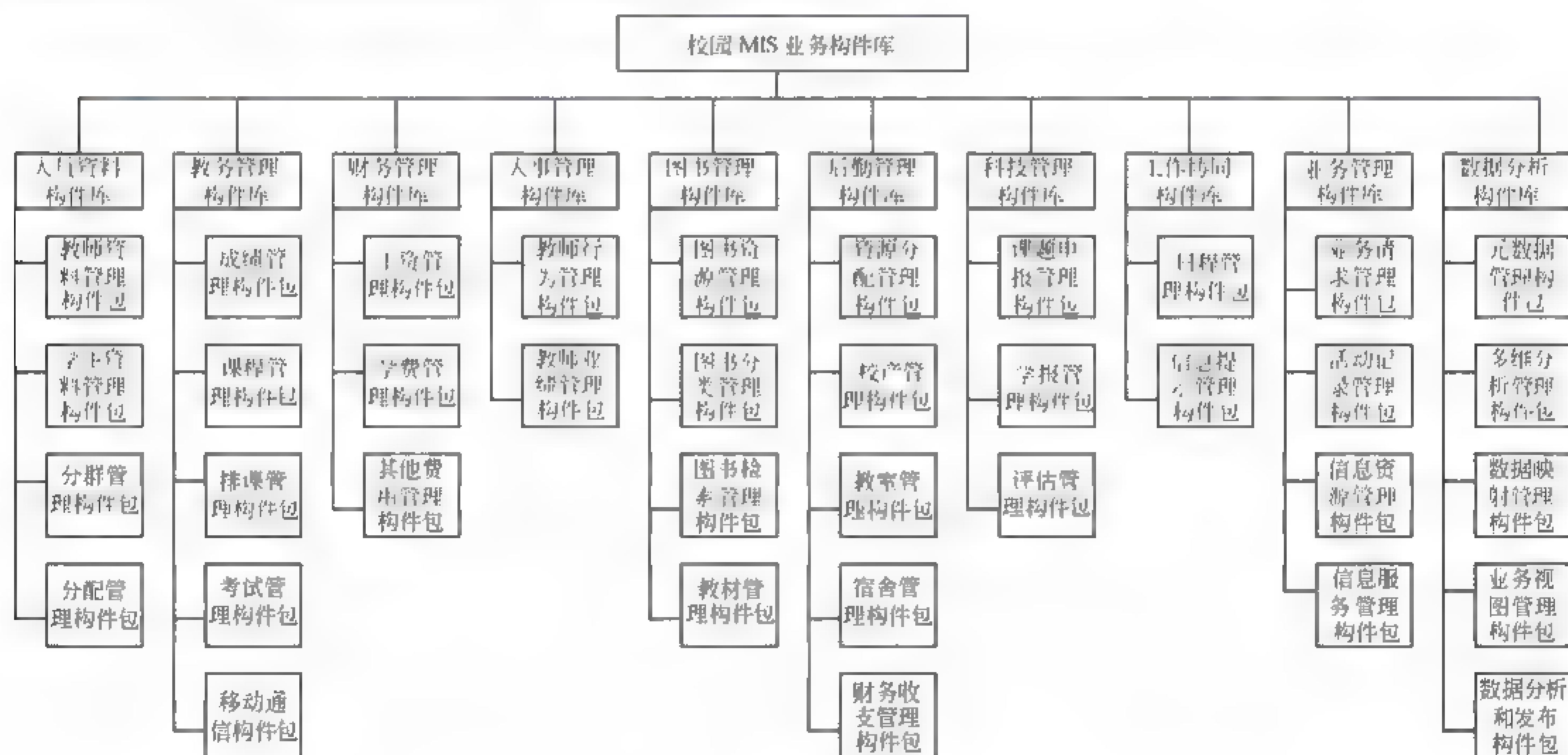

图 2-27　校园业务构件

在图 2-27 中：

① 人员资料构件库：学校内人员资料构件库，对人员基本情况进行抽象，包括教师资料管理构件包，学生资料管理构件包，根据不同的访问管理，又为人员分群管理构件包，分配管理构件包。

② 教务管理构件库：教务管理构件库是以人员资料构件库为基础的，包括成绩管理构件包，课程管理构件包，排课管理构件包，考试管理构件包，移动通信构件包。

③ 财务管理构件库：是指学校财务相关的管理系统构件库，包括工资管理构件包，学费管理构件包，其他费用管理构件包。

④ 人事管理构件库：包括教师行为管理构件包，教师业绩管理构件包。

⑤ 图书管理构件库，这是一个宠大的信息管理领域，这里只给出基本的构件包，包括图书资源管理构件包，图书分类管理构件包，图书检索管理构件包，教材管理构件包。

⑥ 后勤管理构件库：包括资源分配管理构件包，校产管理构件包，教室管理构件包，宿舍管理构件包，财务收支管理构件包。

⑦ 科技管理构件库：包括课题申报管理构件包，学报管理构件包，评估管理构件包。

⑧ 工作协同构件库：包括日程管理构件包，信息提示管理构件包。

⑨ 业务管理构件库：包括业务请求管理构件包，活动记录管理构件包，信息资源管理构件包，信息服务管理构件包。

⑩ 数据分析构件库：包括元数据管理构件包，多维分析管理构件包，数据映射管理构件包，业务视图管理构件包，数据分析和发布构件包。

2.2.1.3 软件体系结构描述方式

软件体系结构描述就是如何表示软件体系结构，目前采用软件体系结构描述语言（ADL）进行形式化描述，其主要表现在对软件体系结构求精、验证、演化和分析。一直以来，软件体系结构描述很大程度上还停留在非形式化的基础上，很大程度上依赖于软件设计师个人的经验和技巧。对软件体系结构的描述通常是采用非形式化的图和文本，不能描述系统期望的存在于构件之间的接口，更不能描述不同的组成系统的组合关系的意义。同时，也让开发人员难以理解，难以进行形式化模拟，难以表现分析设计人员的对软件系统实现的想法，更不具备软件体系结构一致性和完整性检测。但近年来随着XML和语义化的发展和应用，形式化的、规范化的体系结构描述对于体系结构的设计和理解都是非常重要的，如可以通过XML和语义描述实现图形化的UML设计结构实现逻辑化转化，但为了精确描述软件体系结构，通过采用描述逻辑、Petri Net等为基础的数学化描述方式。然而，要实现体系结构设计、描述等的形式化必须先经历一个非形式化的过程，在这发展过程中逐步提取一些形式化的标记和符号，然后将它们标准化，从而完成体系结构设计、描述等的形式化。

1．图形表示工具

图形表示工具是一种简洁易懂且使用广泛的方法，通过采用由矩形框和有向线段组合来表达软件体系结构的描述。在这种方法，流程图代表抽象构件，框内文字为构件名称，有向线段代表辅助各构件进行通信、控制或关联的连接件。目前这种方法在进行分析设计时，还占据主导地位，是因为这种方式易操作、易被理解，但在术语和表达语义上存在着一些不规范和不精确，难以实现语法和语义能表达，也难以实现图形化与实际需求相互转化，因此使得以矩形框与线段为基础的传统图形表达方法在不同系统和不同文档之间有着许多不一致的矛盾。

2．体系结构描述语言（ADL）

ADL是一种形式化语言，它在底层语义模型的支持下，为软件体系结构提供了语法和语义框架。软件系统的结构一般用构件、连接子及它们之间的配置加以描述（构件：计算或数据存储单元或业务功能载体；连接件：用于构件之间交互建模的体系结构构造块及其支配这些交互的规则；体系结构配置：描述构件与连接件的连接图），而支持构件、连接子及其配置的描述语言就是如今所说的体系结构描述语言，典型的ADL包括UniCon、Rapide、 Darwin、

Wright、C2 SADL Acme、xADL XYZ/ADL、ABC/ADL 等。随着软件体系结构研究的不断深入和语义化的发展，研究人员从支持动态、分布、移动系统的建模，针对不同应用领域的建模等，提出了一些新型的 ADL，比如基于可扩展构件模型 Fractal 的 ADL、从结构和行为的角度来描述软件体系结构的 π-ADL、基于多 Agent 系统的 Skwyrl-ADL 等。典型 ADL 特点比较见表 2-11[18]。

表 2-11　典型 ADL 特点比较

ADL	研究单位和代表人物	适用范围	主要设计元素/特点	具备的主要功能
Aesop	美国卡内基梅隆大学软件工程研究所，Davide Garlan 等	同一设计环境中多种软件体系结构风格的应用；特定风格软件体系结构设计环境的快速生成	定义了六种通用对象类型：构件、连接件、端口、角色、表示和绑定；以子类型方式对通用类型进行扩展可定义新类型；每个对象类型用一个 C++类表示，软件体系结构风格信息内嵌在 C++类的代码实现中	外部工具集成；语法制导的类型检查；代码编译；可执行系统的生成；循环、资源冲突、调度可行性检查
C2（）	美国加利福尼亚大学软件研究所，Richard N.Taylor 等	C2 是一种基于构件和消息的体系结构风格，支持大粒度的软件复用和灵活的系统组装；适用于分布式异构环境中；基于消息的图形用户界面应用系统的设计中	主要设计元素包括构件、连接件，以及它们之间的拓扑关系；构件间只能通过连接件相连，具有“受限的可见性”；其异步通知消息和请求消息是构件间唯一的通信方式	体系结构演化；体系结构动态配置；多形式的类型检查；设计决策过程支持；可执行代码生成
MetaH	美国 Honeywell 公司	一种特定领域的 ADL，支持可靠性和安全性要求较高的多处理器实时嵌入式系统的创建、分析和验证，主要适用于航空电子控制软件系统	其语义基于形式化调试和数据流模型；提供预定义的构件和连接件类型，与领域密切相关；构件类型包括事件、端口、子程序、包、监视器和进程等；连接类型包括事件连接、端口连接、等价连接和存取连接等	软硬件绑定；实时调度；可靠性和安全性分析；代码自动生成、编译和链接
Unicon	美国卡内基梅隆大学软件工程研究所，Mary Shaw 等	一种通用类型的 ADL，支持多种常用构件和连接件类型的综合应用	主要设计元素为构件和连接件，构件和连接件类型以枚举方式预定义，并定义了构件类型和连接类型之间的匹配规则；连接件机制内嵌到工具的实现当中	类型检查；编译、链接和可执行系统的自动生成；外部工具集成；进程调度分析
Rapide	美国斯坦福大学，David C.Luchhame 等	基于事件的、复杂、并发、分布式系统的体系结构描述	主要设计元素由接口（相当于构件）和连接规则组成，接口定义包括动作、服务、行为和约束；构件的计算和交互语义通过偏序事件集定义；构件行为约束通过抽象状态和状态转移规则定义	软件体系结构模拟执行；模拟结果分析；软件体系结构的动态配置和代码生成

续表

ADL	研究单位和代表人物	适用范围	主要设计元素/特点	具备的主要功能
Wright	美国卡内基梅隆大学软件工程研究所，Robert J.Allen 等	一种完全形式化的 ADL，将连接件语义进行了显示的形式化表示，支持用户将复杂的交互模式定义成新的连接件类型	主要设计元素包括构件、连接件和配置，构件定义包括端口和计算，连接件定义包括角色和胶水（Glue）；支持用户定义接口类型和风格；以 CSP 作为语义基础	用户自定义软件体系结构风格；风格约束检查；模型一致性、完整性检查；死锁分析
Darwin	英国科学、技术和医药皇家大学，Jeff Magee、Jeff Kramer 等	主要用来对基于消息传递的分布式系统进行描述	主要设计元素由构件组成，构件定义包括提供的服务和要求的服务；没有对连接提供显式的描述，构件之间的交互通过绑定关系表示，以 π-calculus 为语义基础	系统动态配置；代码自动生成和编译
ACME	美国卡内基梅隆大学软件工程研究所，Garlan 等	一种体系结构交换语言，为 ADL 及其工具之间信息的共享和交换提供了一个公共的集成架构	主要包括七个核心设计元素：构件、连接件、系统、端口、角色、表示和重映射	体系结构信息交换和共享
XYZ/ADL	中国科学院软件所，唐稚松等	一种通用类型的 ADL，具有较强的形式化理论基础；能够支持多种设计方法和多种软件体系结构风格的综合运用	主要设计元素包括构件（包括接口和计算）和连接件（接口和交互协议），语义基于时序逻辑语言 XYZ/E，可对设计元素的静态和动态语义进行严格的形式化描述	支持软件体系结构的逐层细化、分析、验证和自动代码的生成；支持用户自定义软件体系结构风格
ABC/ADL	北京大学信息科学技术学院软件研究所，梅宏等	一种通用类型的 ADL，以面向对象分析和设计方法为主导	主要概念包括构件、连接件和体系结构风格，并吸取了面向 Aspect 的开发思想	最突出的特点是支持软件体系结构描述向详细设计和实现的映射，并支持构件的组装
D-ADL	浙江大学计算机软件研究，李赣生等	是一种动态的体系结构描述语言	主要设计元素包括构件、连接件和体系结构风格，且被模型化为高阶多型 π 演算中的抽象（abstraction）类型，系统行为被模型化为进程（process），构件和连接件的交互点则被模型化为通道（channel）	最突出特点就是方便系统变更逻辑的编写、修改和理解；并将动态行为从计算行为中分离出来，显式、集中地表达。由于动态行为可形式化为高阶 π 演算进程，其结果因此能够被预先推导

3．ADL、UML

ADL 的主要目的是帮助软件系统构件人员理解系统，并便于交流。而 UML 是一种可视化的、并能支持分析设计人员从多个视图描述系统，虽然 UML 符号不一定要具备严格的形式化语义，但足以较强的分析能力。在 UML2.0 以前，UML 无法保证与 ADL 转化和紧密衔

接；在 UML2.0 中，增强了对软件体系结构和基于构件的开发的支持能力，增加了诸如结构类、端口、连接件、绑定等体系结构概念，并对构件的语义也做了相应的修改，可以使得 ADL 与 UML 之间相互转换。

4．软件体系结构风格状况

软件体系结构风格是描述某一特定应用领域中系统组织方式的惯用模式。它反映了领域中众多系统所共有的结构和语义特性，并指导如何将各个模块和子系统有效地组织成一个完整的系统。按这种方式理解，软件体系结构风格定义了用于描述系统的术语表和一组指导构件系统的规则。同时，软件体系结构风格是软件工程领域中一个重要的研究问题。总体而言，研究者研究软件体系结构风格的动因体现在三个方面：

（1）研究软件体系结构风格可指导结构设计师选用合适的软件体系结构，降低软件分析设计失败的风险；

（2）一种软件体系结构风格提供特定领域内一种高层的软件复用模板，软件体系结构级的复用，提高软件开发效率。

（3）在软件体系结构的指导下，一种指导软件系统开发的选择软件模式，不同的软件系统可以选择不同的风格加以分析设计，但最终的目的是一样的。

表 2-12 和表 2-13 所示是软件体系结构风格的定义和模式：

表 2-12　软件体系结构风格的定义

定义者	定义描述	关注点
PERRY D E, WOLF A L[19]	从各种相似的、具体详细的体系结构中抽象出的组成元素及其组成关系来表示，比软件体系结构受到的限制更少、更不完全	软件系统的组成元素和元素间的连接关系
BUSCHMANN F, MEUNIER R, OHNERT H，等[20]	软件体系结构风格根据软件系统的结构组织定义了软件系统族：通过构件应用的限制及其与系统结构构建有关的组成和设计规则来表示组成元素和组成元素之间的关系；为一个软件系统及其怎样构造该系统表示一种特殊的基本结构；也包括何时使用它所描述的体系结构、它的不变量和特例以及其应用的效果等信息	区别体系结构风格与体系结构模式两个概念；关注构成软件系统的组成元素及其之间的连接关系；关注组成元素使用的约束条件、表示组成元素间关系的角度以及体系结构风格的适用场合和应用效果
SHAW M, GARLAN D.[21]	是软件系统结构层次上的组织风格，它根据结构组织模式定义一个系统族，未与软件体系结构分开；某些风格还存在一个或多个语义模型，指明如何根据系统各组成成分的属性来确定系统的整体属性	关注的是构成软件系统的组成元素构件和连接关系（连接子）的类型、构件和连接子间的组织方式以及系统的属性
SHAW M, CLEMENTS P[22]	是对构件类型、构件运行时的控制方式与/或构件间数据传递的描述。把一种体系结构风格可看做是在结构上有关构件类型约束及构件间交互约束的一个约束集合，这些约束可以定义一个系统结构族集来满足	不仅关注构件在体系结构中受到的静态约束，更关注运行时在体系结构中的动态约束
Bas , Ctements Kazman[1]	一个程序或计算机系统的软件体系结构所包括一个或一组软件组件（如构件）、软件组件的外部的可见特性及其相互关系	关注软件组件提供的服务、性能、特性、错误处理、共享资源使用状况等

因此，可以将软件体系结构风格描述为软件系统提供了结构、行为和属性的高级抽象，由构成系统的组成元素（如构件）描述，这些组成元素的相互作用，指导组成元素集成的连接以及这些连接的约束组成。

表 2-13 软件体系结构风格的模式（分类）

模式名称（适应领域）	描述	特征	缺点
管道—过滤器风格（系统可划分清楚的模块且模块相对独立，有清晰的模块接口）	每个组件都有一组输入和输出，功能模块称为过滤器，功能模块连接可以看作输入、输出数据间的通路（即管道），且功能之间具备独立性。组件读输入的数据流，经过内部处理，然后产生输出数据流。这个过程通常通过对输入流的变换及增量计算来完成，所以在输入被完全消费之前，输出便产生了	过滤器独立运行构件、处理构件上下连接的过滤器不施加任何限制、不依赖于各个过滤器的先后次序。且支持功能模块的复用，具有较强的可维护性、可扩展性和并发性，支持诸如吞吐量计算和死锁检测等分析	交互式处理能力弱；容易导致系统处理过程成批操作；根据设计要求，设计者需求结合数据传输进行特定处理（也大大的提高了过滤器实现的复杂性）
面向对象风格（多领域）	面向风格模式集数据抽象、抽象数据类型、类继承为一体，使软件工程公认为模块化、信息隐藏、抽象、重用性等原则在面向对象风格下得以充分实现。同时，建立在数据抽象和面向对象的基础上，数据的表示方法和它们的相应操作封装在一个抽象数据类型或对象中。这种风格的组件是对象，或者说是抽象数据类型的实例	适用于数据和功能分离的软件系统中，也适合于问题域模型比较明显，或需要人机交互的系统中，且大多数应用事件驱动风格的系统也常常应用了面向对象风格。因此，面向对象风格具备高度模块性、封装功能、代码共享、灵活性、易维护性和可扩充性	对象间的调用、交互需要标识，这就增加了对象间的依赖性
事件驱动风格（一个系统对外部表现可以从它对事件的处理表征出来）	基本观点是一个系统对外部的表现可以从它对事情的处理表征出来递增性，一个简单的表示方法是为执行系统定义的一些类，另外定义一些作为这些执行系统的容器类（也就是管理系统）。通过是由若干子系统或元素组成的一个整体，并确定系统的目标，使各子系统在某一消息机制的控制下，为这个目标而协调行动，也作为一个整体与环境相适应和协调。同时，在一个系统的若干子系统中，必定有一个子系统起着主导作用，而其他子系统则处于从属地位，并且对于任一个系统和系统内的任一元素，都有一个事件收集机制和一个事件处理机制，通过这种机制与周围环境发生作用和联系	事件驱动风格非常适合于描述系统族，在属于同一族的任何系统中，系统的高级管理子系统的描述是完全类似的，便于重用。这时，高级管理系统牢牢的掌握着控制权，又因为各同级子系统一般不直接发生关系，因此实现并发处理和多任务操作。同时，基于事件驱动风格的系统具有良好的可扩展性，设计者只需为某个对象注册一个事件处理接口就可以将该对象引入整个系统，也不影响其他的系统对象定义了包含执行子系统和管理子系统的类层次结构。简化客户代码。使基于事件驱动的系统设计更具有一般化	构件削弱了自身对系统计算的控制能力；不得于数据共享；使各个对象的逻辑关系变得更加复杂
分层风格（适应功能层次抽象一相互之间低耦合的系统）	一个分层系统采用层次化的组织方式构建，系统中的每一层都承担为上层提供服务和为下层提供功能函数，即每一层为上层提供服务，并作为下层的客户端。在分层风格的体系结构中，一般内部的层只对相邻的层可见。层之间的连接器（Conector）通过决定层间如何交互的协议来定义	这种风格支持基于可增加抽象层的设计。允许将一个复杂问题分解成一个增量步骤序列的实现。由于每一层最多只影响两层，同时只要给相邻层提供相同的接口，允许每层用不同的方法实现，同样为软件复用提供了强大的支持。因此，具有在系统设计过程中逐级抽象、较好的可扩展性和支持软件复用等优点	不是所有的系统都适合分层风格来描述；对于抽象出来的功能具体应该放在哪个层上也是设计者难把握的难点

续表

模式名称（适应领域）	描述	特征	缺点
数据共享风格（适用特定的领域）	通常情况下，数据共享风格包含中央数据单元构件和一些相对独立的构件集合两个截然不同的功能构件。这是由于共享信息交互方式的差异导致了控制策略不同，通过控制策包括基于传统数据库型数据共享风格的系统和基于黑板型数据共享风格的应用系统，其中黑板型由知识源、黑板数据结构和控制三部构成，它对于无确定性求解策略的问题比较有用，在专家系统中应用比较广泛	数据共享风格控制原则的选取产生两个主要的子类。若输入流中某类时间触发进程执行的选择，则数据共享是一个传统型数据库；系统中的组件通常包括数据存储区，以及与这些存储区进行交流的进程或处理单元，而连接器则是对于存储区的访问。这类系统中，数据处理进程往往并不直接发生联系，它们之间的联系主要是通过共享的数据存储区来完成的。这种现象非常类似于在独立组件架构中的情况。另一方面，若中央数据结构的当前状态触发进程执行的选择，则数据共享是一黑板系统。因此，数据共享风格具有解决问题的多方法性、可更性、可维护性、可重用知识源、支持容错性和健壮性	数据共享风格也存在测试困难、效率低、开发成本高、缺少对并行机支持和不能保证有好的求解方案等缺点
解释器风格（虚拟机风格）（适用特定的领域）	基于解释器风格的系统核心在于虚拟机；通过包括被执行的伪码和解释引擎两部分，其中伪码由需要被执行的源代码和解释引擎分析所得中间代码组成，解释引擎包括解释器和解释器当前的运行状态	通常情况下，解释引擎从被解释的模块中选择一条指令；然后基于这条指令，引擎更新虚拟机内部的状态；并反复执行这一过程，直到满足要求为止。因此，解释器风格具有这些优点：在方法规则比较简单的情况下，解释器风格工作很好；易于改变和扩展方法，易于实现方法；可以用多种操作来完成一个需求	解释器风格也存在无法解释复杂的方法规则、应用范围比较狭窄，以及在方法规则比较复杂，方法的层次变得无法管理，这时，系统中需要包含许多表示方法规则的类
反馈控制环风格（适用特定的领域）	对一个对象或过程进行控制，意味着设法使这个被控对象或过程的功能或特性有效的达到所期望的目标，但为了成功设计一个控制系统，必须事先知道被控对象具有的性质和特征，同时还须了解和掌握这些性质和特征随环境等因素变化的情况。控制工程是一个专业领域，应用时也是独立于各种应用功能。在采用反馈控制环风格时，当从单纯的控制领域中抽象出来时，一般需要引入动态系统的概念	通过反馈控制环风格的动态系统在实现信息处理和传输一个功能单元时，由系统起因和由此引起的时间上的效果分别作为系统的输入量和输出量，这样定义的系统具有目标作用、信息处理、闭环和开环控制过程的共同性。当然控制论也可以应用于软件体系结构的创建	通常只适应于特定领域，且把握需求对象和过程的性质和 特征
异构风格的集成（适用于需求明确特定的领域）	各种系统构建模式之间不仅有联系，而且在很多情况下它们往往是配合使用的，这样的系统可以称为复合型系统，所构建模式就称为异构风格集成	面向一个实际系统，很难判断它究竟是 A 型，还是 B 型，亦或者是 C 型，单纯的把它归到任何一型都是不科学的	异构数据源难控制；异构模式间集成点难获取

续表

模式名称（适应领域）	描述	特征	缺点
REST 风格（基于Web的应用）	在2000年，则Roy Thomas Fielding在其博士论文首次提出REST风格。是针对Web应用而设计，它是为了降低开发的复杂性和提高系统的可伸缩性	网络上的所有事物都被抽象成资源；每个资源对应一个唯一的资源标识符；通过通用的连接器接口对资源进行操作；对资源的各种操作不会改变资源标识符；所有的操作都是无状态的；强调中间媒介的作用（如代理服务器、网关）等	难获得准确的资源，一量标识失效，资源就无法找到
SOA 风格（应用广泛）	是一种面向服务体系结构的新型软件体系风格，可以有效解决分布处理所带来的松散耦合、语言无关和平台无关等问题，一般采用Web服务进行实现	具有松散耦合、语言无关、平台无关等优势，并且SOA最大限度的包含了以前相关技术，使已有的IT投资得到了保护和重用	分析设计难度较大，对需求把握难以实现
云计算风格（应用广泛）	是一种基于互联网的计算方式，通过这种方式，共享的软硬件资源和信息可以按需提供给计算机和其他设备。整个运行方式很像电网。可以通过浏览器等软件或者其他Web服务来访问，而软件和数据都存储在服务器上。云计算通常可以包含基础设施即服务（IaaS），平台即服务（PaaS）和软件即服务（SaaS）三类云。云计算服务通常提供通用的通过浏览器访问的在线商业应用，软件和数据可存储在数据中心。且具有较好的敏捷性、可扩展性、可维护性、可靠性等	基于虚拟化技术快速部署资源或获得服务；实现动态的、可伸缩的扩展；按需求提供资源、按使用量付费；通过互联网提供、面向海量信息处理；用户可以方便地参与；形态灵活多样，聚散自如；减少用户终端的处理负担；降低了用户对于IT专业知识的依赖	云安全
正交风格（适用于特定领域）	是一种以垂直线索为基础的层次化结构，一般由组织层和线索构成，组织层是由具有相同抽象级别的构件组成，线索是相对子系统的特例，是由分布在不同层并有调用关系的构件组成	正交软件体系结构风格就是将软件功能正交分解，按功能的正交相关性，垂直分割为若干线索，线索又分为几个层次。对于大型的且比较复杂的软件系统，线索还可以划分为更低一级的线索，形成多级正交软件体系结构	不易划分组织结构和线索
RIA 风格（适用于特定领域）	是一种提升用户体验的桌面应用程序，它具有反应性、交互性强且与Web应用程序零部署、易升级、易操作等优势，从而简化了Web应用程序的用户交互，增强用户的体验	强交互性，支持丰富的UI组件；局部数据更新，即通过客户端计算可直接实现对用户请求的响应；所有的内容在一个界面中，易于实现多步骤处理；RIA集成XML特性，简化异质系统通信，方便数据存取；具有较好的平台无关性	多适应于界面处理

2.2.1.4 软件体系结构方法

软件体系结构是对软件系统整体组织结构和控制结构的刻画，包括系统中各计算单元（构件）的功能分配、各单元之间的高层交互说明（连接件）以及软件体系结构的约束。但对于不同的软件需求，所采用的软件体系结构方法也是有所不同的；虽然不同的软件需求可以采

用同一软件体系结构方法给予实现，但所带的复杂度是无法估量的。因此，针对不同的软件需求，需要采用不同的软件体系结构方法，以达到有效的、快速的完成软件构架。同时，软件体系结构方法就是对软件体系结构风格的一种具体体现，一种实际的构架。表 2-14 是对目前有影响的软件体系结构方法分类：

表 2-14　软件体系结构方法分类

软件体系结构方法	描述	原理	特征
面向普适计算的方法[23]	在普通计算和对等计算空间中进行适应性抽象建模的基础上，建立名为 UbiArch 的普适计算软件体系结构，UbiArch 支持软件实体的按需加入应用、主动适应环境的行为模式，实现了软件适应能力的高层重用。同时，与构件等现有成熟软件技术的紧密结合保证了 UbiArch 的可实践性	建立以适应性为中心建立普适计算空间抽象模型；将该抽象模型映射到构件等成熟软件技术之上。通过对已有技术的扩展来实现集中体现适应性的 Join/Adapt 操作语义；从概念视图、运行视图和开发视图三个维度对 UbiArch 软件体系结构表达	在构件等成熟软件技术基础上，通过引入感知和行为构件模型，增加行为驱动引擎和应用加入部件扩展元层容器功能，使得软件实体可以主动适应环境；通过建立应用系统到普适计算空间的映射机制、定义适用于不同场景的自主单元加入模型，使得软件实体可以按需加入应用。最终验证表明：该软件体系结构方法具有有效性和通用性
以决策为中心的方法[24]	根据体系结构层次设计的决策抽象和问题分解原则、一种以决策为中心的体系结构设计方法。该方法从决策的视角对体系结构进行建模，并通过一个从导出体系结构关键问题到对体系结构方案决策的过程完成设计，还在其中实现了候选体系结构方案的自动合成以及设计决策与理由的捕捉	以决策为中心的软件体系结构方法是根据体系结构层次设计的决策抽象和问题分解原则分析设计的。第一、反映体系结构设计实质的决策抽象原则。第二、降低体系结构设计困难的问题分解原则。第三、通过问题、方案、决策、理由等概念对设计决策进行建模，从而得到体系结构设计决策的元模型；并且体系结构设计决策模型与制品模型（包括结构模型、行为模型等）共同构成了完整的体系结构模型。第四、以体系结构决策为中心的体系结构设计过程包括包含问题导出、方案发掘、方案合成、体系结构决策、理由捕捉五个主要活动；该过程的输入为软件需求规约，输出为软件体系结构描述	以决策为中心的软件体系结构方法在体系结构设计过程中实现了候选体系结构方案的自动合成，使得架构师只需完成相对简单的问题方案发掘，降低了直接发掘体系结构方案的难度和人工探寻体系结构方案解空间的复杂性
面向动态演化的方法[25]	面向动态软件体系结构的在线演化方法是设计并实现了一种运行时刻的软件体系结构元模型，将原先运行时刻不可见的体系结构设计信息具体化为显式的体系结构实体，并与系统实现及系统规约之间保持因果关联。元模型的演化可通过反射实现对运行系统的修改和对规约的更新，所有演化行为都在良定义的体系结构元模型的指导下规范地进行	该软件体系结构方法设计阶段的体系结构规约被具体化为内置于运行系统的可编程的体系结构元模型，在体系结构元模型的基础上自顶向下开发的实现系统与元模型维护着因果连接的关系。通过反射运行时刻对体系结构元模型的修改一方面驱动了软件系统的演化，另一方面实时更新了与之关联的体系结构规约。因此，首先分析了软件系统的动态特性和面向 DSA 的软件设计、实现、演化架构，然后实现了面向 DSA 的在线演化实现方法	该软件体系结构方法通过可编程的体系结构元模型将系统中的构件的交互解耦，实现基于体系结构上下文的松耦合；利用反射技术和编程模型中的设施使所有演化都在运行时刻体系结构的指导下进行可保证系统演化前后的一致性、完整性和追溯性

续表

软件体系结构方法	描述	原理	特征
以通用的适应性方法[26]	该体系结构方法通过对适应性软件体系结构的基本特性分析，抽取适应性软件体系结构的通用框架，建立基于多视图建模理论的集成化适应性软件体系结构参考模型，给出多视图模型的演化与映射规律。基于元建模和图转换理论，提出模型映射一致性算法。并建立了适应性软件体系结构支撑环境。适应性软件体系结构适用于复杂软件系统，特别是网络环境下大型、开放式软件系统的开发和实施	该软件体系结构方法通过对适应性软件体系结构的基本特性分析，提出适应性软件体系结构的通用框架，进一步抽取出适应性软件体系结构参考模型。该参考模型独立于具体的计算环境，具有普遍适用性。并指出该参考模型的组成以及软件生命周期阶段中模型间的映射关系和模型演化过程进行了深入的研究，并建立了相应的集成化开发环境，在工具上为基于适应性体系结构的软件开发、部署与运行维护提供支持	该方法在通用的适应性软件体系结构风格方面力求抽取出适应性体系结构的公有特性和通用参考模型。并对适应性体系结构的形式化分析、视图一致性维护及动态有效性判定，以及基于适应性体系结构的自动求精等方面的研究
形式规约生成的方法[27]	使用 LOTOS 描述实时系统需求规约，通过建立 LOTOS 规约到 UML-RT 模型的模型转换，提出一种基于形式化规约生成软件体系结构模型的方法	该软件体系结构方法是通过 MDE（model-driven engineering）开发模式相应的支撑平台 AMMA 所提供的技术，通过元建模的方法实现 LOTOS 规约到 UML-RT 模型的转换。第一、基于 AMMA（ATLAS，Atlantic data systems）平台的 KM3（kernel metameta model）元模型体系，通过元建模抽象出 LOTOS 与 UML-RT 的 KM3 元模型，实现 LOTOS 与 UML-RT 的同构化。第二、利用平台的模型转换语言 ATL（ATLAS transformatio-n Language）针对 LOT-OS 元模型和 UML-RT 元模型构造转换规则.通过将对应的实例模型进行相互转换，实现在 MDE 下 LOTOS 规约到 UML-RT 模型的转换	该软件体系结构方法从软件需求规约自动变换到 SA 模型是采用基于 LOTOS 规约生成的 SA 模型来保留了形式化语义，所以通过这种方法建立的 SA 模型比通过自然语言规约建立的 SA 模型更为精确，不存在二义性；并且 LOTOS 规约到 SA 模型的转换都有元模型之间映射与之对应，因此需求和 SA 设计之间保持了良好的可追踪性
面向方面的方法[28]	该方法是在分离软件系统中的核心关注点和横切关注点的情形下，通过引入面向方面软件开发的思想设计的一种面向方面软件体系结构模型方法。并指出了该模型的三个基本构成单元，即构件、连接件和方面构件	面向方面的软件构架方法一般认为面向方面软件体系结构在传统软件体系结构基础上增加了方面构件（Aspect Component）这一新的构成单元，通过方面构件来封装系统的横切关注点。最终在软件体系结构的构架上实现关注分离	该方法通过建立构件、连接件和方面构件三个基础构成单元构架的软件体系结构具有一定的理论意义和实用 价值

2.2.1.5 软件体系结构可靠性

随着软件系统规模的不断膨胀、信息量日益增多，架构对软件可靠性和开发成本的影响也越来越大。作为在软件生命周期早期保障软件质量的重要手段之一，软件体系结构评估技术和可靠性分析是软件体系结构研究中的一个重要组成部分，也是保障软件质量有效的方法

（软件质量是指软件对预期的一系列质量属性组合的满足程度），并且是降低开发成本，提高软件可用性、可信性重要保障手段之一。同时，与当前可信软件也有直接关系。而根据 1983 年美国 IEEE 计算机学会将软件可靠性定义为：

（1）在规定的条件下，在规定的时间内，软件不引起系统失效的概率，该概率是系统输入和输出的函数，也是软件中存在的缺陷的函数。

（2）在规定的时间周期内，在规定的条件下，程序执行所要求的功能的能力。

与此同时，基于软件体系结构的层次来度量软件系统的可靠性也是非常重要的，这是由软件体系结构自身的特点以及它在软件开发过程中的特殊地位决定的。同时，对软件体系结构进行可靠性分析，能够尽早发现软件系统中可能存在的问题，以便通过特定的方法和技术来改善系统的可靠性[29]。

当前，基于构件的软件系统的可靠性模型分为两种：一种是将基于构件的软件系统当成一个整体，并不考虑其内部构成；另一种是考虑构件软件系统的各个构件的可靠性以及软件系统的结构。前一种做法因为没有考虑各构件的可靠性和体系结构对整个软件系统的影响，所以其应用受到一定限制。采用上面的模型对软件体系结构的可靠性属性进行度量存在以下不足[29]：

（1）模型在计算各构件模块可靠性和构件之间的转换概率时，依赖于专家的经验数据。

（2）软件的每个用例执行的概率是不一样的，执行概率较小的用例所涉及的软件部分对软件体系可靠性属性的影响相应也较小。

（3）软件部件失效后的严重性是不一样的，有些较为简单的构件失效后会导致比较严重的后果，应该对部件的失效后果区别对待。

（4）软件体系结构中的连接件失效概率没有考虑。

针对软件体系结构的可靠性分析目前最重要的方法之一就是采用评估手段来实施，并采用软件体系结构的质量属性（一个组件或一个系统的非功能性特征。依照国际标准 ISO/IEC 9126-1，共有六种特征：功能性，可靠性，可用性，有效性，可维护性和可移植性）来描述，其主要目的是为了识别体系结构设计中的潜在风险（即可靠性），验证系统的质量需求在设计中是否得到了体现，预测系统的质量并帮助开发人员进行设计决策等功能，并通常采用 Petri 网进行建模和描述。

2.2.1.6　软件体系结构基本应用

采用一个实习实训系统为例来简要分析软件体系结构的基本应用方法。

1．实习实训系统的简介及目标

实习实训系统是数字化校园信息系统的重要组成部分，是动态管理和跟踪即将毕业学生实习实训的一种有效的信息化手段，因此实习实训在校园数字化信息系统中占相当大比重。实习实训信息和数字化校园信息之间是相互关联不可分割的，二者的有机结合能充分发挥出各方面信息的价值，有效地为各项业务及决策服务，使实习实训系统的应用效果得到充分发挥。

而搭建一个实习实训系统的目标是建立稳固、强大的企业级通信基础设施；利用信息技术发展的契机，加强校园数字化建设；各种指令上传下达畅通，响应迅速；加强协作，提高全体教职员工整体工作效率；充分利用信息资源，提高决策效率和准确度；加强与学生交流，提高学校应变能力。

2. 技术特点分析

实习实训系统有其自身的特点，它的信息量大、信息种类繁多、参与人员多，需要不同层次的协作，更有一些信息代表着指令和命令，需要有安全、保密的保障和可认证性。因此：

（1）群件技术：实习实训系统中的信息种类多，可能包括文字、表格，图形图像，甚至包括声音动画等多媒体信息，需要多种软件进行协调处理，实习实训系统的文档多为复合的文档，信息的传递更需要与功能强大的网络互动技术结合。它能使不同的信息的处理协调进行，使人们在不同的地域和时间协同工作，促进群体交流和资源共享，充分提高群体工作效率。

（2）工作流技术：实习实训系统中，一个工作的完成需要多个人员参与，参与的角色、时间、阶段和方式也各不相同，因此需要有强有力的工作流技术作保证。要能自动寻找路由传递文档，供相应人员进行批阅；跟踪传递中文档的状态，对工作进行统计和督办；企业级与跨企业级的文档分发；文档的组织与链接等。

（3）安全机制：实习实训系统中的信息既有上传下达的指令与命令，又有不同人员的个人信息。因此在传送和保存中要有严格的安全机制作保证。数字签名：确保信息来自发出信息的人，以防信息被仿冒和篡改；私人密钥加密：保证信息传输和到达后只有指定的人员才能看到；存取权限控制：从数据库、文档、乃至区段字段级的加密，使有存取权限的人员才能阅读或修改相应的内容。

（4）移动计算：移动计算技术使人们离开办公室时，如：在家中，同样可以连入校园网络进行办公，如同在办公室一样，可以大大地提高工作效率，提供学校竞争能力。

（5）支持SQL标准：使实习实训系统能对关系数据库进行访问，另外从实习实训审批流程中的数据记入管理信息系统的数据库。

（6）与Internet的互连：信息技术的发展使人们可以通过Internet进行信息交流、信息发布。校园内部网络不再是孤立的、封闭的系统，通过防火墙与Internet互连，使校园网与外界能及时地交流信息。

3. 系统的要求

（1）网络互动：该模块提供了非常强大的交互、互动功能。通过该功能不仅可以实现教师与学生间的互动交流，还可以让学生通过该平台向老师求助，老师再向学生回答问题；不仅可以持久化数据，并可以跟踪问题。

（2）文档数据库系统：不能可以存储实习实训的关键数据和学生与教师信息外，还可以存储学生上传的附件。

（3）交互式的Web服务器：完全支持Internet标准，可以发布或浏览Web信息。所有的应用系统信息都可以在安全机制的控制下，动态发布给Internet或Intranet用户，同时通过Internet/Intranet收集信息。

（4）集成性：最终用户使用浏览器。在相同的、唯一的界面下，用户可以获得实习实训信息。这种集成能力是在软件平台一级获得的，而不是通过二次开发“强加”上去的。这大大减少开发的工作量，提高了应用系统的可靠性。而且对最终用户来说，不需要在几个客户端软件或是应用程序之间、几个服务器系统或数据源之间来回切换。

（5）开发快捷、实施容易：实习实训系统的推出与使用是投资得以回报的关键。

（6）管理容易、机制完整：由于采用单一的软件系统，所有的应用开发都建立在统一的平台之上，主要的维护工作集中在应用管理一级，管理工作的难度与工作量大大降低。对于

安全性管理、系统可靠性管理等重要工作，管理人员可以使用后台管理平台的管理手段。

（7）保有费用低：应用系统的保有费用包含了购买费用、管理维护费用、应用开发实施费用、系统集成费用等等。由于采用了功能强大的单一产品，本系统的总体保有费用在各方面都会降低。

4．系统分析

图 2-28 是整个实习实训系统的功能组成用例图，系统由管理人员、指导老师和学生三个角色、五个功能模块组成。

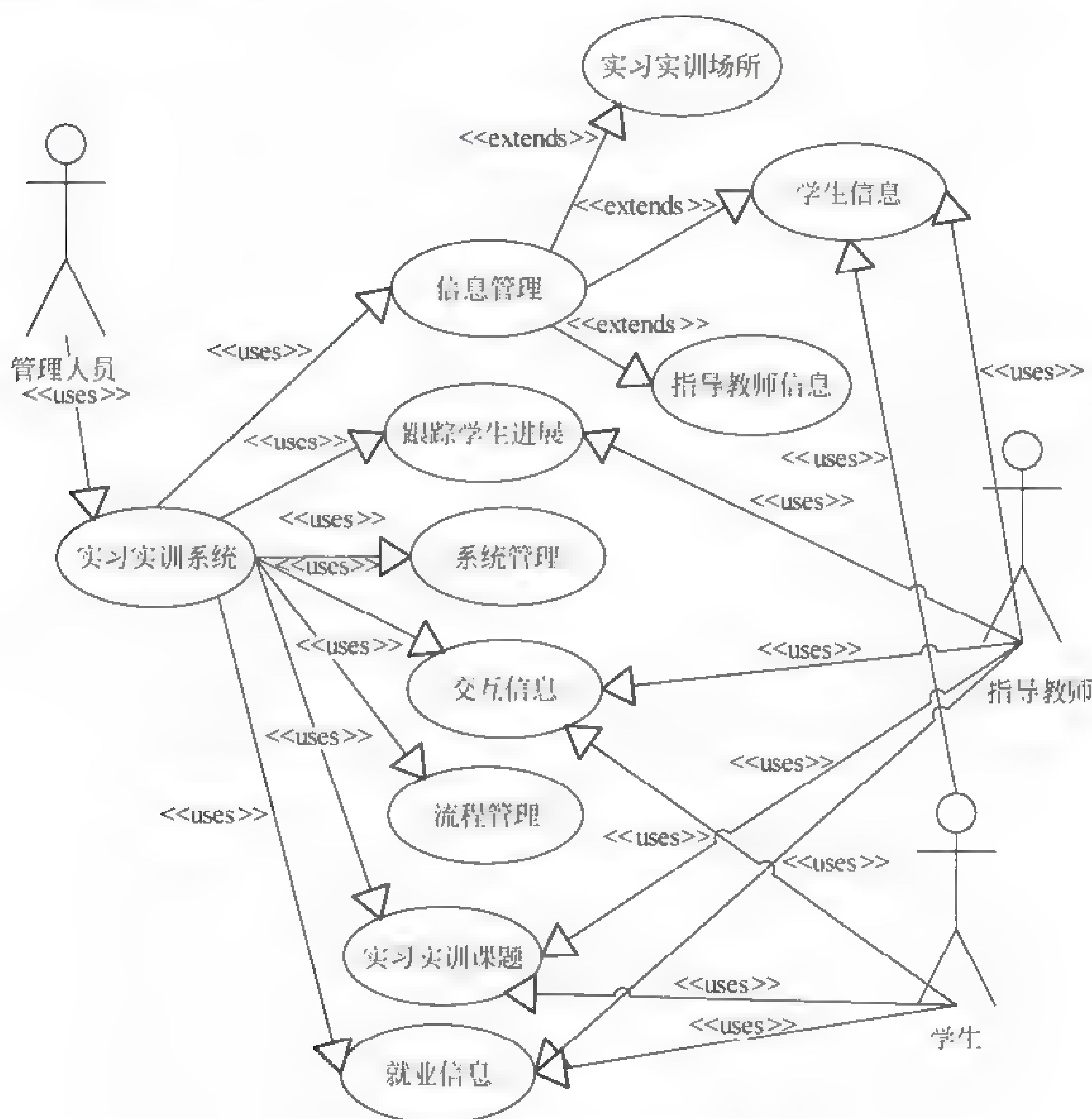

图 2-28　实习实训系统功能结构图

2.2.2　软件层次结构

软件层次结构是目前软件逻辑构架最通用性的方法之一，但与其他类型的层次结构在表达功能和描述形式是不同的，如计算机网络 ISO/OSI 参考层次模型就是表达网络结构和设施的。而软件层次结构主要用于软件构架，并且这些层都负有独立的职责，但每个层与相邻层之间存在松散耦合性。下面主要介绍以 J2EE 为核心的设计模式的分层、面向构件的分层模式、面向服务的分层模式和面向云的分层模式。

2.2.2.1 J2EE 分层模式

J2EE（Java 2 Platform Enterprise Edition）是当前信息化解决方案最主要方法之一，这是因为为搭建具有可伸缩性、灵活性、易维护性的商务系统提供了良好的机制。主要表现在能保留现存的 IT 资产、高效的软件研发、支持异构环境、稳定的可用性。并且 J2EE 服务器以容器的形式为所有的组件类型提供后台服务，这就为 J2EE 程序的编写十分简单，业务逻辑被封装成可复用的组件更为容易。同时，J2EE 平台由一整套服务（Services）、应用程序接口（APIs）和协议构成（这些 API 接口包括如下所示），并对开发基于 Web 的多层应用提供了功能支持，图 2-29 所示是 J2EE 的体系结构，图 2-30 所示是 J2EE 分层核心模式。

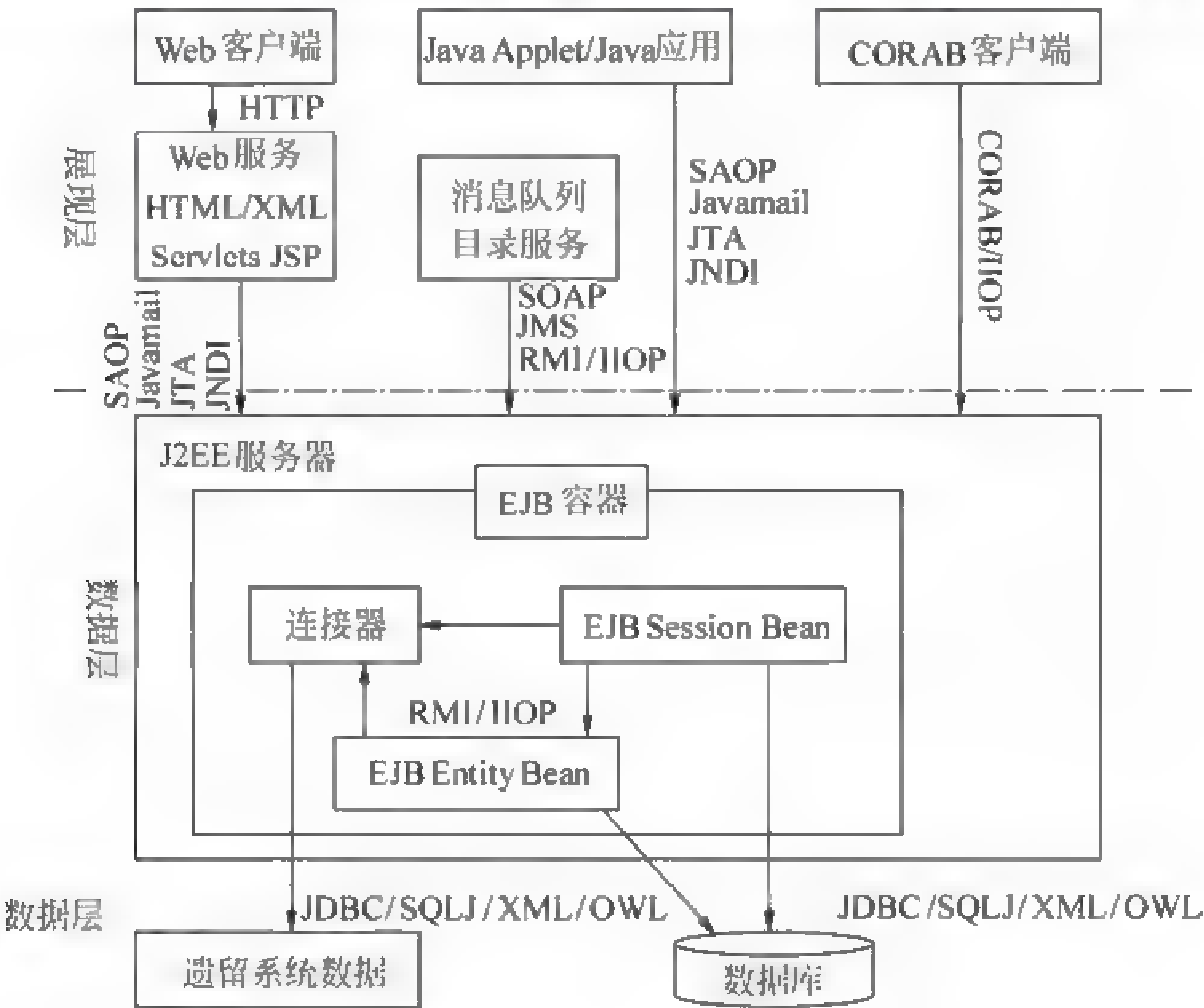

图 2-29 简化了的 J2EE 体系结构

下面简要介绍组合 J2EE 的核心部件：

（1）JDBC（Java Database Connectivity）：JDBC API 为访问不同的数据库提供了一种统一的途径，与 ODBC 具有一样的特性，JDBC 对开发者屏蔽了一些细节问题，另外，JDCB 对数据库的访问也具有平台无关性。

（2）JNDI（Java Name and Directory Interface）：JNDI API 被用于执行名字和目录服务。它提供了一致的模型来存取和操作企业级的资源，如 DNS 和 LDAP，本地文件系统，或应用服务器中的对象。

（3）EJB（Enterprise JavaBean）：J2EE 技术之所以赢得某体广泛重视的原因之一就是 EJB。它们提供了一个框架来开发和实施分布式商务逻辑，由此很显著地简化了具有可伸缩性和高度复杂的企业级应用的开发。EJB 规范定义了 EJB 组件在何时如何与它们的容器进行交互作用。容器负责提供公用的服务，例如目录服务、事务管理、安全性、资源缓冲池以及容错性。但这里值得注意的是，EJB 并不是实现 J2EE 的唯一途径。正是由于 J2EE 的开放性，使得有

的厂商能够以一种和 EJB 平行的方式来达到同样的目的。

图 2-30　J2EE 核心分层模式

（4）RMI（Remote Method Invoke）：正如其名字所表示的那样，RMI 协议调用远程对象上方法。它使用了序列化方式在客户端和服务器端传递数据。RMI 是一种被 EJB 使用的更底层的协议。

（5）Java IDL/CORBA：在 Java IDL 的支持下，开发人员可以将 Java 和 CORBA 集成在一起。它们可以创建 Java 对象并使之可在 CORBA ORB 中展开，或者它们还可以创建 Java 类并作为和其他 ORB 一起展开的 CORBA 对象的客户。后一种方法提供了另外一种途径，通过它 Java 可以被用于将新的应用和旧的系统相集成。

（6）JSP（Java Server Pages）： JSP 页面由 HTML 代码和嵌入其中的 Java 代码所组成。服务器在页面被客户端所请求以后对这些 Java 代码进行处理，然后将生成的 HTML 页面返回给客户端的浏览器。

（7）Java Servlet：Servlet 是一种小型的 Java 程序，它扩展了 Web 服务器的功能。作为一种服务器端的应用，当被请求时开始执行，这和 CGI Perl 脚本很相似。Servlet 提供的功能大多与 JSP 类似，不过实现的方式不同。JSP 通常是大多数 HTML 代码中嵌入少量的 Java 代码，而 servlets 全部由 Java 写成并且生成 HTML。

（8）XML（Extensible Markup Language）：XML 是一种可以用来定义其他置标语言的语言。它被用来在不同的商务过程中共享数据。XML 的发展和 Java 是相互独立的，但是，它和 Java 具有的相同目标正是平台独立性。通过将 Java 和 XML 的组合，可以得到一个完美的具有平台独立性的解决方案。

（9）JMS（Java Message Service）：JMS 是用于和面向消息的中间件相互通信的应用程序接口（API）。它既支持点对点的域，又支持发布/订阅（publish/subscribe）类型的域，并且提供对下列类型的支持：经认可的消息传递，事务型消息的传递，一致性消息和具有持久性的

订阅者支持。JMS还提供了另一种方式来对应用与旧的后台系统相集成。

（10）JTA（Java Transaction Architecture）：JTA定义了一种标准的API，应用系统由此可以访问各种事务监控。

（11）JTS（Java Transaction Service）：JTS是CORBA OTS事务监控的基本的实现。JTS规定了事务管理器的实现方式。该事务管理器是在高层支持Java Transaction API（JTA）规范，并且在较底层实现OMG OTS specification的Java映像。JTS事务管理器为应用服务器、资源管理器、独立的应用以及通信资源管理器提供了事务服务。

（12）JavaMail：JavaMail是用于存取邮件服务器的API，它提供了一套邮件服务器的抽象类。它不仅支持SMTP服务器，也支持IMAP服务器。

（13）JTA（JavaBeans Activation Framework）：JavaMail利用JAF来处理MIME编码的邮件附件。MIME的字节流可以被转换成Java对象，或者转换自Java对象。大多数应用都可以不需要直接使用JAF。

在图2-30中：

（1）客户端层：该层代表访问系统的人员，应用程序，或系统的客户端。它是整个系统的对外接口，可以是Web浏览器,Java应用程序（Swing），Java Applet，WAP，其他设备或者是批处理程序。也包括UI表现和UI设备，实现用户间的交互。

（2）表示层：该层封装了用来服务访问本系统的所有客户端的表示层逻辑；该层解释客户端的请求，提供单次登录、集中登录，实现会话管理、内容创建、内部格式和传送，控制对业务的访问（权限检查），构造客户端的回复（response），以及把回复传递给客户端；一般由Sevlet,JSP驻留在该层。

（3）业务逻辑层：该层提供业务服务，包括业务数据和业务逻辑。通常应用程序的大多数业务处理集中在本层。同时它管理事务，一般EJB驻留在该层。

（4）集成层：该层负责与外部系统和外部资源通信，它有多种方式，如与数据库连接使用的JDBC、XML等。

（5）资源层：该层包括业务数据源和外部系统资源，如Oracle数据库,JMS server，其他遗留系统等。

2.2.2.2 面向构件的分层模式

面向构件技术主要注重组织级可重用的层次粒度和关注点分离组装；而目前面向构件思想是以分布式中间件技术（包括企业应用集成中间件）来表现的，并通过构件虚拟机（是一种构件基础设施，面向构件应用可以运行于上）为载体实现面向构件组装。其主要注重以构件库为中心的关注点分离（主要包括重用和组装），以及不同层次的粗粒度和各种联接接口的松散耦合度，而各构件中的业务、任务数据通过本体接口用XML数据总线的联结适配器进行描述、发现服务，然后完成业务流程处理。同时，按照构件关注点分离的区别和层次粗粒度的大小可分为业务逻辑和服务构件、业务构件、构件系统，其业务构件是最大粗粒度的可重用构件单位，关注点分离是保持构件自治性，散松耦合度是实现平台、语言的无关性。而构件系统是由一系列业务独立的构件子系统组成，其每个构件子系统由一个或一系列业务构件组装而成。构件库是作为整个基于构件的整个系统的可利用资源支撑，如图2-31所示。最后到达以人为中心的业务流程生成、经营管理方式。其建立一种角色构件来完成访问控制，

然后角色通过配置文件静态或动态分派到各构件中去实现访问控制：

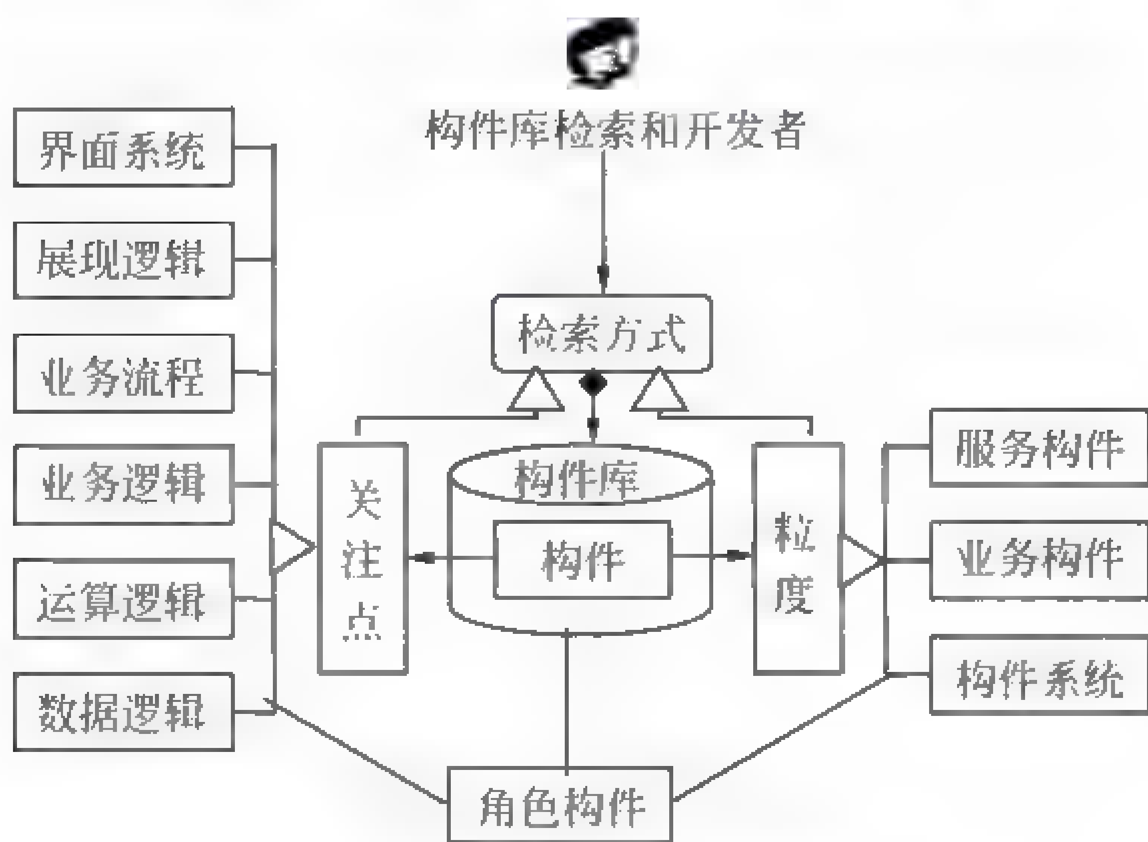

图 2-31　构件组织图

面向构件的软件研发与设计，同样采用层次结构来实现，按照文献[15]定义把构件分层由上向下分为业务流程层、用户界面层、构件层、逻辑层、运算层和数据层，且每个是以具体的构件为基本元素，通过数据总线实现各个层之间的数据传递，以提高各个层次数据的扩展能力。图 2-32 是面向构件的分层模式：

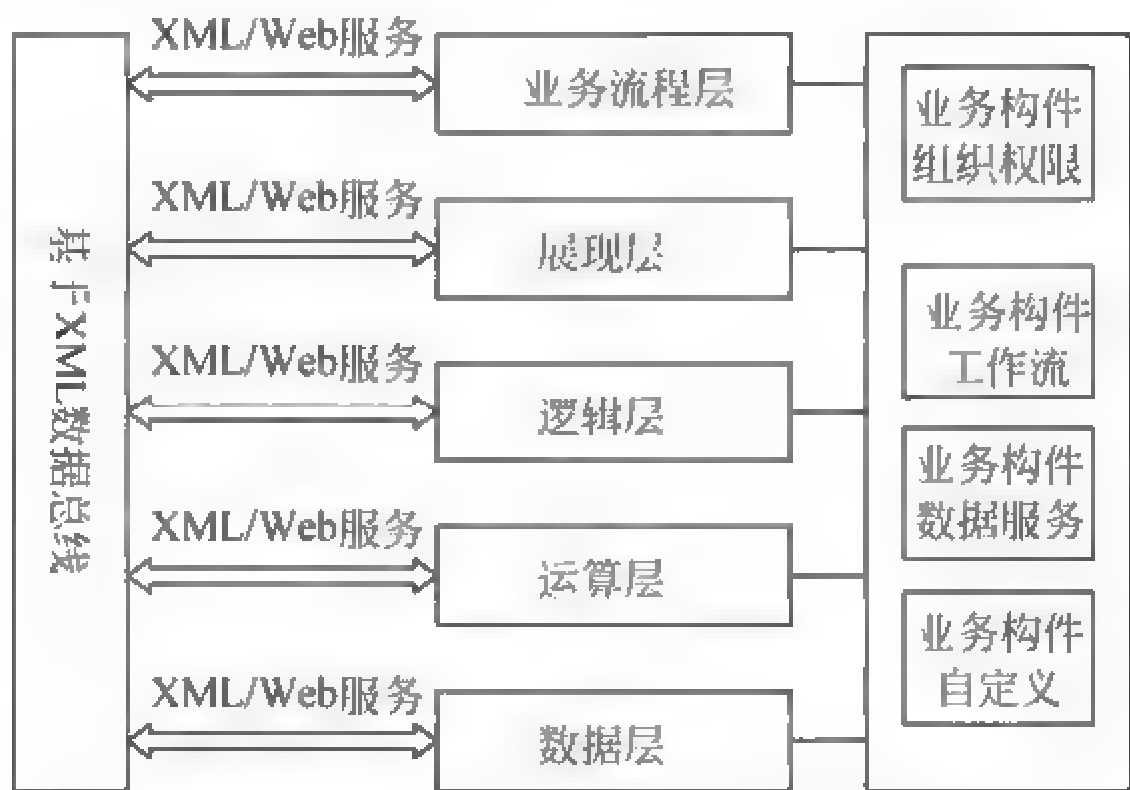

图 2-32　面向构件的分层模式

在图 2-32 中[15]：

（1）业务流程层：根据用户的业务流程定义，完成人工或自动的业务活动；这些业务流程定义来自于用户需求变化。

（2）层现层：是连接用户界面与业务处理的中间层次。且层现构件将用户提交的数据传递给相应的逻辑层构件进行调用，根据调用的返回，不规则定位到另一个用户界面，并把业务处理的返回数据传递到相应页面上。

（3）逻辑层：是实现应用逻辑的处理过程，实现方式是通过构件组装环境将许多运算构件结合在一起实现复杂的业务处理过程，以提高开发、测试和维护效率。也使得应用的逻辑与具体代码实现了有效的分离。

（4）运算层：是对计算机处理操作的构件化封装。

（5）数据层：主要实现业务数据访问逻辑与企业应用系统数据源的分离，并同时在关注点的驱动下提供统一的数据访问接口，这样降低了对数据访问的依赖和松散耦合关系。提升了企业应用在数据层次的扩展能力。

2.2.2.3 面向服务的分层模式

由于服务计算具有独特的可伸缩性、跨平台、跨语言、高松散耦合性等多种优势，现已有逐渐成为了一种面向服务体系结构（SOA）的软件构件模式。SOA是一种分布式的软件模型，主要组件由服务、动态发现和消息组成，目前通常由Web服务、SCA和SDO进行表达。因此，SOA一般由服务提供者、服务消费者和分布服务描述、服务注册中心组成，通过URI直接使用服务描述，并通过注册中心查找服务描述、绑定和调用服务，而服务代理（Service Broker）提供和维护了服务注册中心。目前通常用Web服务来实现SOA是一种较好的选择。图2-33是SOA概念模型：

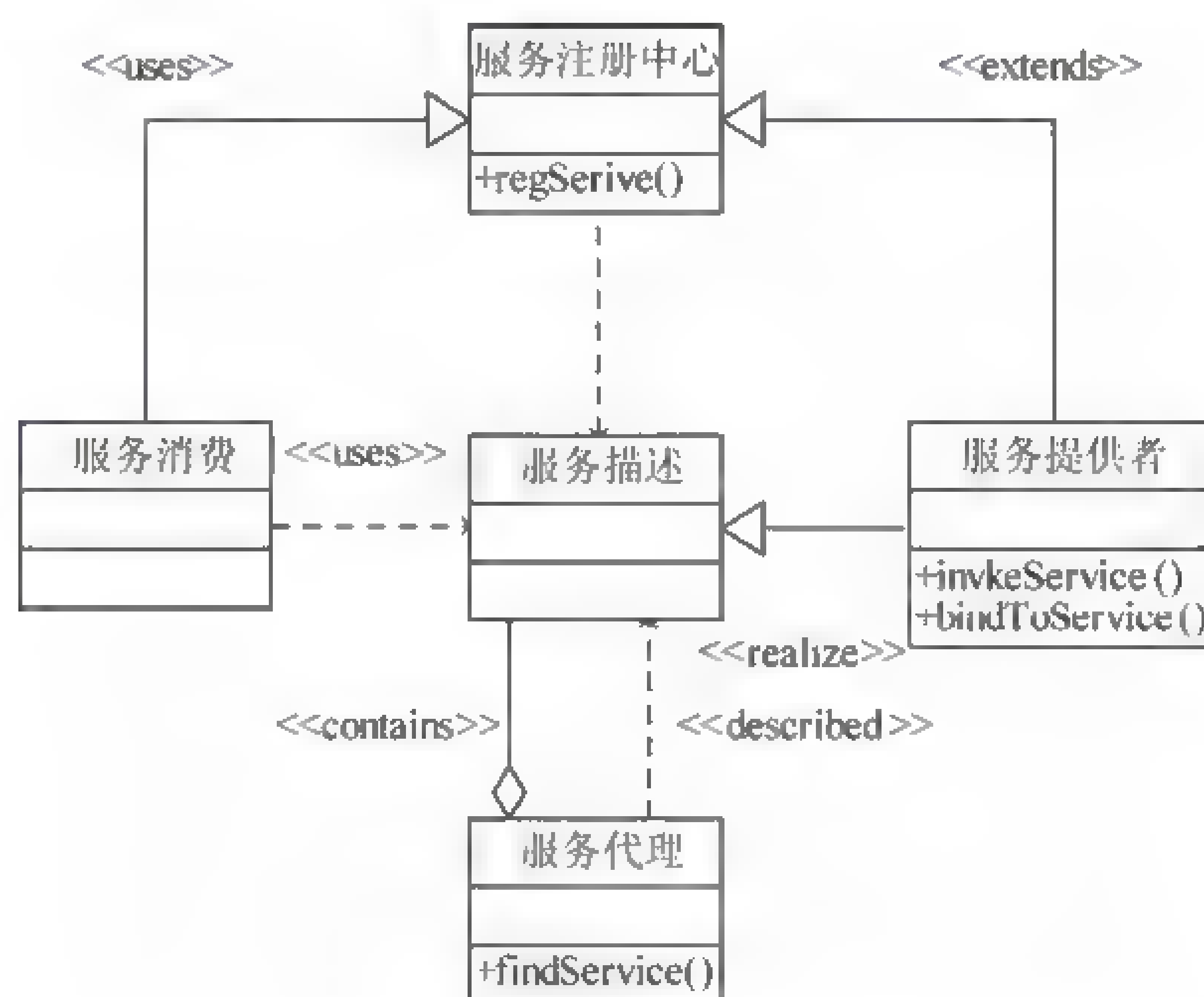

图2-33 SOA架构概念模型

由图2-33所示可知，SOA定义了一系列模式和指导方针来创建松耦合、依赖业务的服务来描述、实现、绑定之间关系分离，这样就为IT系统的新业务组装提供了灵活性，并通过服务上下文来鉴别、制定、监视、管理和控制动态组装系统的实现。

SOA实行的基础是组合服务提供者和服务消费者各个业务功能和流程，实现复杂的业务应用程序和流程，其相对于粗粒度的业务组件被作为服务公开，也将IT资产构造为一系列可复用的服务，而且这些服务是松散耦合、与平台和实现无关；但SOA解决方案设计为服务组合的编排和排列，并为基于构件的组装提供良好的接口和契约进行连接。在2006年10月由OASIS组织制定和发布了SOA参考模型1.0版本中，使用了概念、关系和语义来描述服务，并使用了Capabilities这一方式进行描述SOA，图2-34是所示SOA的参考模型。

其具体的构架实现是通过目前SOA的七层层次结构来完成的，分别为：操作系统层、企业组件层、服务层、业务过程合成层、表现层、集成层（ESB）和服务质量层（QoS）。图2-35所示是SOA层次结构。

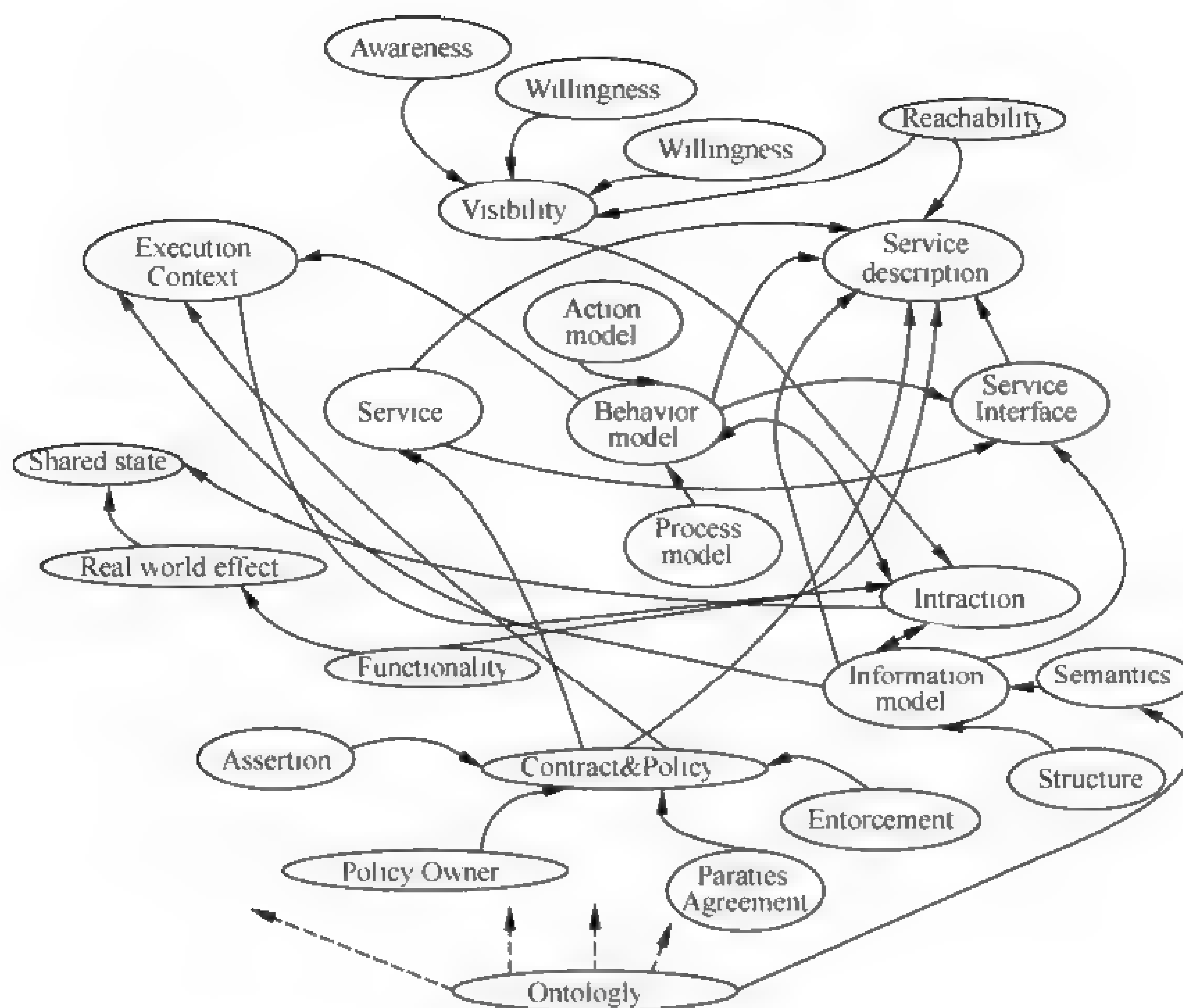

图 2-34　SOA 参考模型

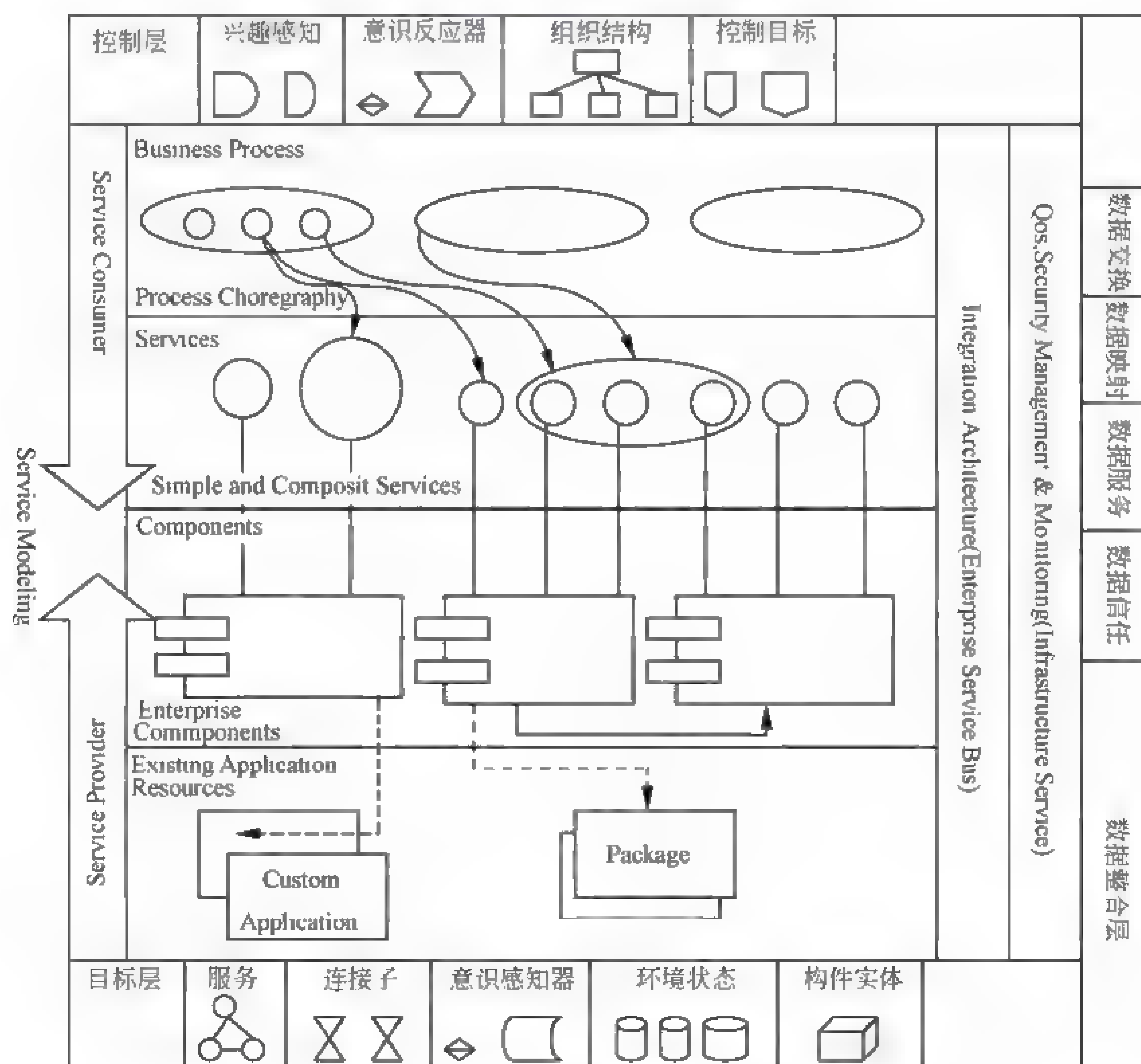

图 2-35　SOA 层次结构（面向服务的分层模式）

（1）服务质量层（QoS）：提供一组监视、管理和维护的功能，包括系统安全、性能和可用性等的 QoS 能力，通过 WS-Management 和其他相关的技术标准来实现，达到 SOA 服务质量的标准。

（2）集成层（ESB）：是一组基于 XML 的语义推理的服务集成模式总线，其通过引入一系列可靠的性能集合来完成 WSDL 或语义绑定，包括智能路由、协议中介、服务地址和其他转化机制，通常称为 ESB，且 ESB 为集成提供位置的独立机制。

（3）表现层（Face）：这一层主要由表现技术来实现，如 Portlets、WSRP、Ajax 等，然后通过表现层应用程序接口或表现层访问接口来利用 Web 服务；这样，SOA 将用户接口从组件中分离出来，提供访问路线到服务或合成服务的端到端的解决方案。

（4）业务过程合成层（BPC）：该层是用于合成编排服务，以形成一个一个的业务流程，具体的通过配合、编排、服务绑定来实现业务流程，并且可以作为单独的应用程序共享，同时，通过查找业务驱动实现一个一个的业务功能。

（5）服务层（Service）：这一层主要实现业务选择和管理暴露的服务，因此，可以被发现或者静态绑定、调用，然后编排合成到服务中，以形成业务流程。同样，可以通过服务暴露来获取企业范围组件、业务单位特定组件、特定的服务构件组装机制，并且以服务描述的形式具体化接口子集，包括各种连接适配器。这样，企业组件就可以使用适配器暴露的功能在运行时提供服务实现，然后通过接口公开的服务描述独立存在或合成服务。

（6）企业组件层（EC）：该层主要负责实现功能和保持暴露服务质量（QoS）的企业组件组成，包括企业和业务单元级、组织级支持的企业资产受管理和控制的集合，相关的业务功能可以是基于构件的或容器的，如实现组件、负载均衡、高可重用性、快速组装和工作量管理的构件库或应用服务器。

（7）操作系统层（OS）：这一层主要实现遗留系统与新系统集成，主要表现打包原有的应用程序，如现有的 ERP、CRM 等，然后暴露相关接口，参与服务。

2.2.2.4 面向云计算的分层模式

云计算目的是实现网络环境可伸缩性和可靠性，它是一种新型、基于 Internet 的分布式计算方法，是对并行计算、分布计算和网格计算进一步扩展，以构建下一代计算机网络，现正在如火如荼的在各领域进行深入研究和初步应用。目前相关云研究的体系结构主要从应用层、虚拟机层和物理层三个层次去研究云计算架构。图 2-36 所示是一种云的层次结构。

（1）Users：包括云应用的开发人员、用户、终端用户、云系统管理者。他们通过 Web 服务（SOAP）和 REST 实现各自的需求，或通过各自其他访问模式实现各自需求；但它们最终通过服务应用控制平台完成业务逻辑操作。

（2）Application：指具体的应用方法，通过 HTTP(s)、Servlet、WSDL、Semantics 访问和识别云统一访问接口（UCRI），并由 UCRI 实现寻址中断访问 Internet 公开的资源。

（3）Resources：包括 SaaS 提供商、PaaS 提供商、安全验证、各类软件服务的业务构件、服务引擎、目录管理等，以及基于 QoSaaS-Driven 的驱动约束机制。还包括虚拟机监控、寻址中断调度、寻址中断工作流程和服务管理。

（4）Virtual Machine（VM）：所有的虚拟机都由开源件 Eucalyptus 管理、分派、计算。

（5）Physical Layer：即 IaaS，支持云计算的硬件平台，这些硬件分布于不同地点、不同

网络环境中。

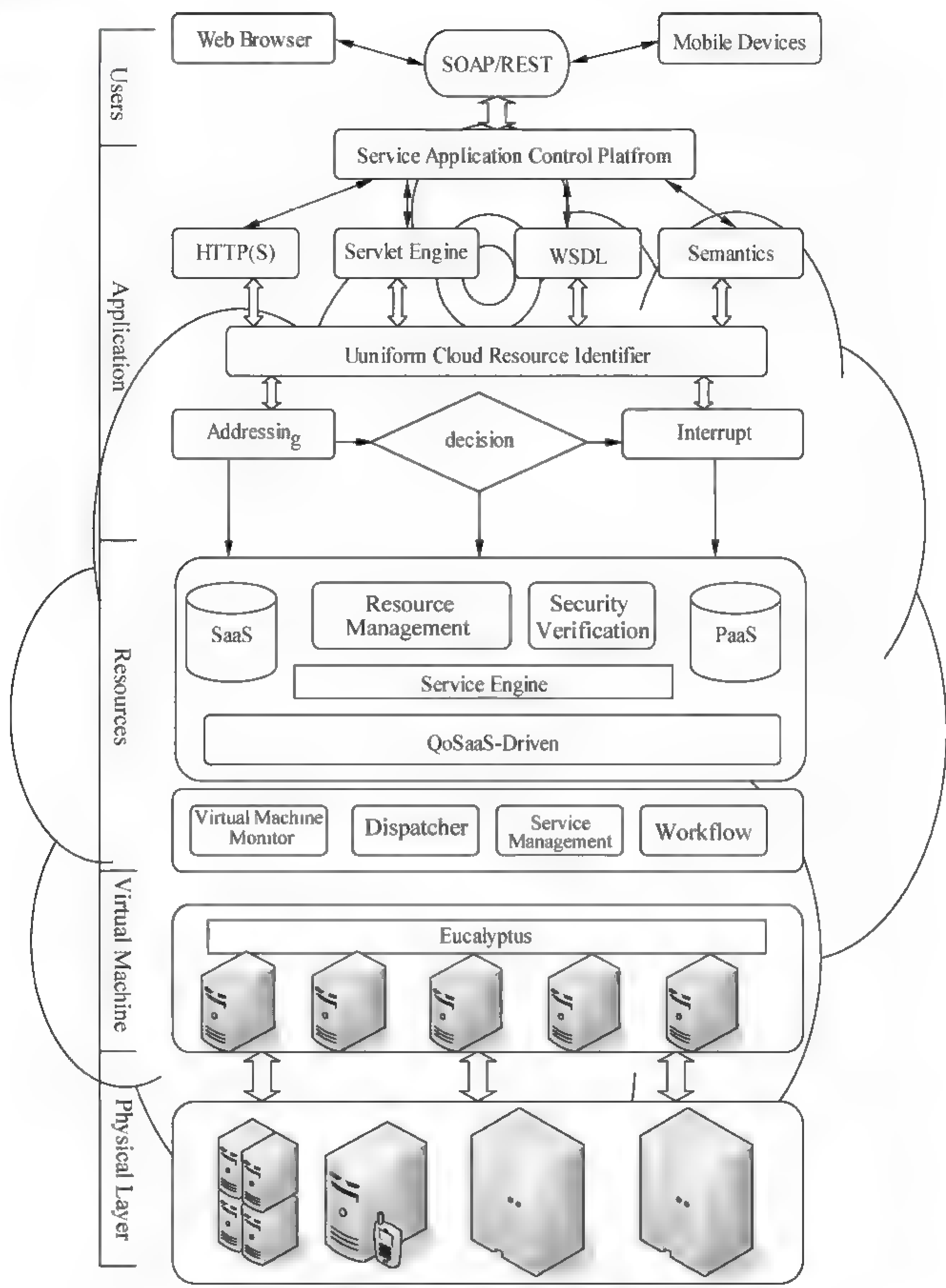

图 2-36　面向云计算的分层模式

以上四个小节分别介绍了目前软件构架领域中常见的四种分层模式，并给出每种分层结构的示意图。虽然每种分层模式都是采用层堆积而成，但所表达的含义和各层间的功能说明和逻辑描述是不一样的。表 2-15 所示是各分层模式的区别和比较。

表2-15 各分层模式的比较

模式种类	作用对象	方法与载体	代表性语言	关注点
J2EE分层模式	类、对象	方法、继承	C++、Java、C#	抽象、封装
面向构件的分层模式	构件、构件库	层次粒度、关注点分离	无或XML	复用、组装
面向服务的分层模式	服务、软件实体	WSDL、UDDI、OWL-S	XML、OWL-S	复用、组合
面向云计算的分层模式	服务、虚拟机	IaaS、PaaS、SaaS	XML、OWL-S	按需求生成、复用

2.2.3 软件中间件构架方法

中间件（MIDDLEWARE）是提供系统软件和应用软件之间连接的软件，即中间件是建立在操作系统、网络和数据库之上，应用软件的下层，总的作用是为处于自己上层的应用软件提供运行与开发的环境，帮助用户灵活、高效地开发和集成复杂的应用软件，图2-37是中间件基本体系结构图。其目的以便于软件各部件之间的沟通，特别是应用软件与系统软件间的逻辑通信。而中间件在现代信息技术中的应用框架有Web服务、面向服务的体系结构（SOA）等，应用领域也是相当广泛的，如能源、电信、金融、银行、医疗、教育等行业软件，可以说中间技术现已深入到各领域。只要是一些公共的功能存在，都可以以中间件的形式来表达，从而提高软件可重用率，简化繁琐的代码编写工作，降低了面向行业的软件的开发成本。同时，当前一般认为在商业中间件及信息化市场中主要存在微软阵营、Java阵营、开源阵营，而阵营的区分主要体现在对下层操作系统的选择以及对上层组件标准的制订。其中微软阵营的主要技术提供商来自微软和与之相关的商业伙伴，Java阵营则来自IBM、Sun、Oracle、金蝶（Kingdee Apusic）及其合作伙伴，开源阵营则主要来自诸如Apache、SourceForge等组织的共享代码。而中间件分类，大致可分为六类：终端仿真/屏幕转换中间件、数据访问中间件、远程过程调用中间件、消息中间件、交易中间件、对象中间件。当然中间件技术的蓬勃发展是离不开标准化的，这是因为标准的创建有助于融合不同阵营的系统；目前越来越多的标准被三大阵营共同接受并推广发展。最终使得中间件技术的发展方向朝着更广阔范围的标准化，功能的层次化，产品的系列化方面发展。

也可以把中间件定义为“连接软件组件和应用”的计算机软件，它包括一组服务，以便于运行在一台或多台机器上的多个软件通过网络进行交互。并且中间件技术所提供的互操作性，有效推动了分布式体系架构的演进，而该架构通常用于支持分布式应用程序并简化其复杂度，它包括Web服务器、事务监控器和消息队列软件。同时，中间件正在向需求变迁、可成长性、适应性、可管理性和高可信性等方面发展和更新。

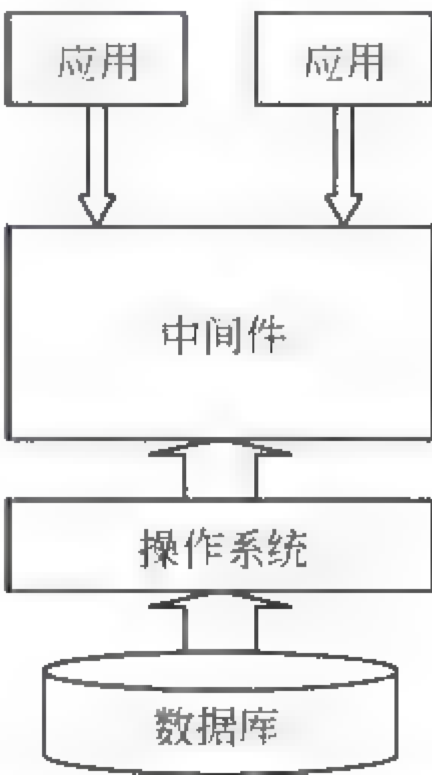

图2-37 中间件基本体系结构

该中间件优点有：

（1）建立灵活的基础架构；

（2）实现业务流程整合，提高业务灵活性；

（3）降低IT复杂性，降低业务整合成本；

（4）简化整合软件开发，改善基础架构管理，加速实现价值；

（5）充分利用现有的技术资产，提高透明度；

（6）改善IT服务的管理和质量，建立能够满足法规要求的IT架构；

（7）提高现有 IT 投资的价值；

（8）提高信息可用性、质量和价值。

中间件能够屏蔽操作系统和网络协议的差异，为应用程序提供多种通信机制；并提供相应的平台以满足不同领域的需要。因此，中间件为应用程序提供了一个相对稳定的高层应用环境。然而，中间件服务也并非“万能”，也存在一些不足：

① 中间件所应遵循的一些原则离实际还有很大距离，如多数流行的中间件服务使用专有的 API 和专有的协议，从而使得相关应用软件建立于单一厂家的产品，并造成来自不同厂家的实现的软件很难互操作。

② 有些中间件服务只提供一些平台的实现，从而限制了应用在异构系统之间的可移植性，使得应用开发者在这些中间件服务之上建立自己的应用还要承担相当大的风险。

2.2.4 轻量级的软件构架方法

重量级方法具有很多规则、惯例、和文档，正确地遵循它们需要反复训练和时间；而轻量级方法仅具有很少的一些规则和惯例，或者说，这些规则和惯例遵守起来很容易。下面从原则、过程和技术三个方面分析轻量级的软件构件方法。

2.2.4.1 轻量级的软件构架方法基本原理

1. 原则

（1）简单性：即面向轻量级的软件构架过程应该刚好能完成所需的软件构架工作，并让简单性渗入到所有的工作中去，使开发人员应该尽量使用最简单的方法解决问题，且能用工具构建一个清晰、简洁的解决方案。

（2）修补漏洞：许多开发方法可能不鼓励在过程中进行重构或变更，因为这些行为不直接用于产生客户代码；轻量级开发要求可以自由地修补太复杂或充斥 bug 的代码。

（3）自动化单元测试：应该优先编写测试用例，广泛的单元测试来改善客户的体验，并提高代码的设计水平。

（4）使用短开发周期并积极调动客户参与其中：即剔除不必要的软件构架工件来简化软件构架周期。如果已经顺利得到了客户的参与，那么很多的功能规范就会变得越来越没必要，这样也容易使得客户满意这种模式，这是因为稳步提高了客户的业务价值。

2. 过程

（1）专注现场客户和代码，而不是其他设计技巧。

（2）简化需要的文档。即为了满足需要，宁可使用电子表格中的一行来描述，也不使用令人困惑的用例图。

（3）只做足以完成工作的设计工作。不要对设计或性能过分忧心忡忡，使自己陷入绝境。

（4）为了开发，努力进行简化并保证至少每天都集成所构建的程序，必要时能进行重构。

（5）自动化测试。

3. 技术

（1）依赖注入：最新一代容器器称为轻量级容器，它们使用一个共同设计原理：依赖注入。对这个简单思想来说，这是一个复杂的术语。依赖注入将对象和它所依赖的东西交给第三方，然后第三方创建所有对象并将它们绑在一起。

（2）AOP（Aspect Oriented Programming）：在不修改源代码的情况下，AOP 可以通过预

编译方式和运行期动态代理模式来实现程序动态、统一添加业务逻辑功能。AOP 实际是 GoF 设计模式的延续，而设计模式孜孜不倦追求的是调用者和被调用者之间的解耦，因此 AOP 可以说也是这种目标的一种实现。

（3）透明持久性：持久性也是建立在较简单的编程模型之上。透明持久性框架通过配置而不是编写代码，来给应用程序添加持久性。因为大多数应用程序是面向对象的，并且访问一个关系数据库，所以一些专家断言：软件构架最终将进入对象关系映射的时代。目前发现的顶级持久性解决方案是 SolarMetric 的 Kodo JDO 和 Hibernate。在后面的文章中我将详细比较这些解决方案，以及其他轻量级解决方案。

2.2.4.2 轻量级构架实现

轻量级构架仍是采用层次结构将不同功能属性的框架融入不同的层次结构中，目前在面向信息化建设领域中，最常用的就是轻量级开源软件框架 SSH(i)[Struts + Spring + Hibernate（iBatis）]，图 2-38 所示就是 SSH（SSH 详细分析见后续章节）层次结构图。

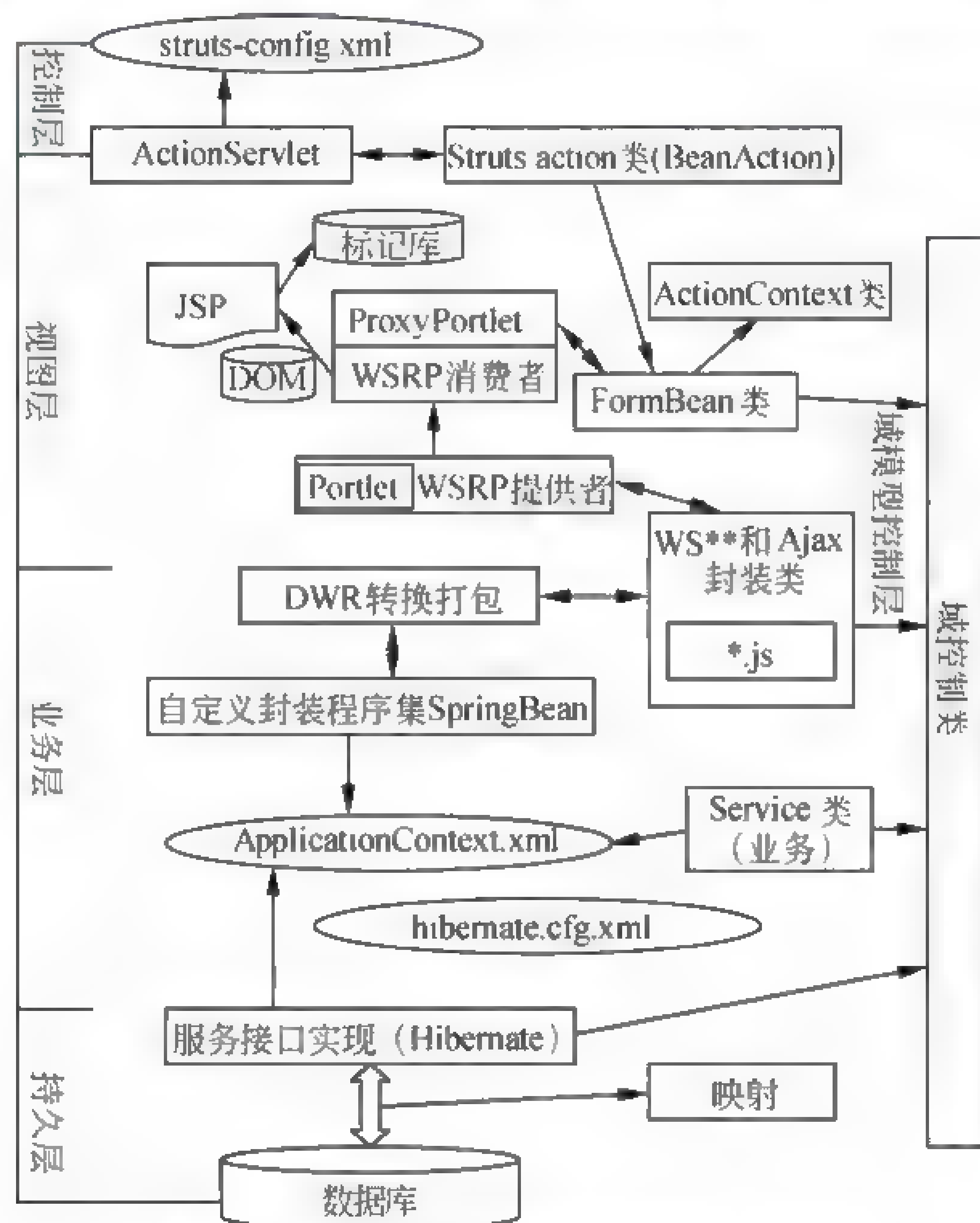

图 2-38　面向轻量级的 SSH 层次结构图

在图 2-38 中：

（1）表示层。表示层是基于 MVC（Model-View-Controller）的 Struts 框架以及 Ajax 技术，同时 Struts 与 Spring 都有很好的衔接，它是目前使用最多、最好的 Web 框架之一，其中模型（Model）是应用程序主体部分，表示业务数据或者业务逻辑；视图（View）是应用程序中用户界面相关的部分，是用户看到并之交互的界面；控制器（Controller）是根据用户的输入，控制用户界面数据显示和更新模型对象状态。模型的角色可以通过 Spring 的业务对象类来实现，控制器可以用 Servlet 来完成。而 Ajax 技术是为了解决现今基于 Web 的应用系统所遇到

同步等待问题的一种 Web 技术，其实 Ajax 并不是一门新技术，而是多种技术的整合体，它采用客户端脚本（JavaScript）与 Web 服务器交换数据完成实时刷新页面，采用 XML 完成异步处理。其组成技术为：

① XHTML 和 CSS 提供了基于标准的表示，其中 CSS 可以表示为：选择符 {属性：属性值}；

② 文档对象模型（Domcument Object Model）提供动态显示和交互；

③ XML 和 XSLT（XSL Transformations）提供了数据交换和操纵；

④ XMLHttpRequest 提供异步数据检索，XMLHttpRequest JavaScript 对象实现实时异步处理；

⑤ JavaScript 把每一样东西绑定在一起。在客户端使用 JavaScript 进行交互处理数据，但编写 JavaScript 代码复杂、结构性差等缺点，而且很多应用程序是 JavaBean 程序，因此出现了 Driect Web Remoting（DWR），它是 Apache 许可下的开源码的 Java 库，用于构建基于 Ajax 的 Web 应用程序，其目的是向开发人员隐藏 Ajax 细节，在服务器端使用普通的 Java 对象（POJO），如基于 Spring 的 Java 对象。DWR 就可以把 POJO 动态地生成 JavaScript 代理函数供客户端调用，DWR 其组成是由一个 Java Servlet 和 utils.js、engine.js 两大部分组成。然后视图通过 DOM 解析后用 JSP 进行显示，在控制器与模型间用 DWR 来实现向 JavaScript 转换，然后通用视图进行表现，图 2-39 是 Ajax 与 DWR 运行时序图。

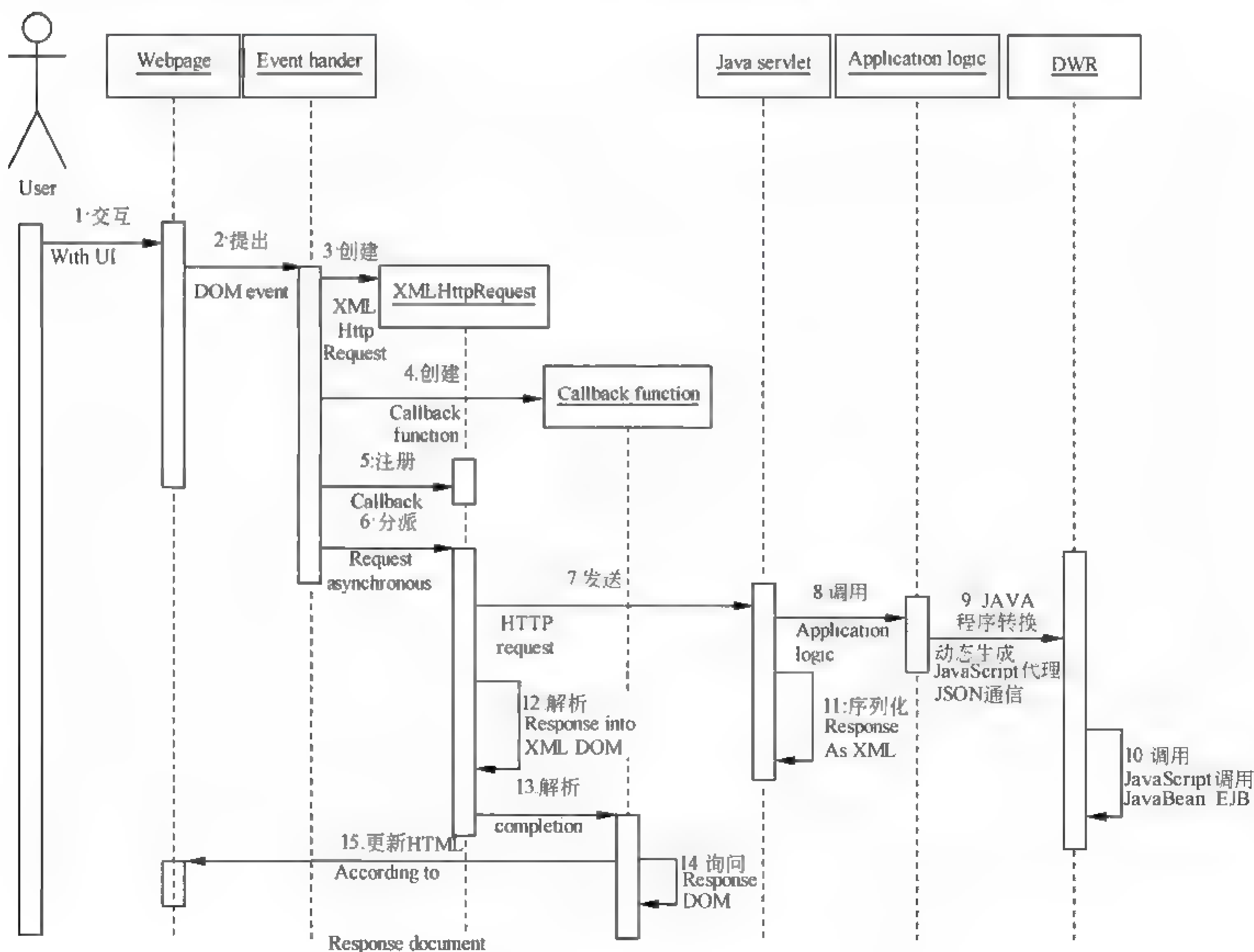

图 2-39　Ajax 与 DWR 运行时序图

（2）业务层。采用 Spring 框架来实现业务层，它是一个开源框架，并与 Struts 已很好的结合，其通过内置 Struts Plug-in 在两者之间提供了很好的结合点，同时，Spring 与 Web 服务也实现很好的联结。而框架的主要优势之一就是其分层架构，分层架构允许选择使用任何组件，并为 J2EE 应用程序的开发提供了集成的框架。并通过 DWR 完成向 JavaScript 转换供在浏览器上显示。其主要是构建业务对象（BSO），用来执行程序逻辑、调用持久层。同时使用 Spring 框架来管理该层，能给应用程序带来极大的灵活性和松散的耦合度。首先建立业务适配器实现 Spring 框架中 Bean 注册，在实现 Bean 时必须注重接口与具体类型的松散耦合度，因为 Spring 框架是采用面向方面机制（AOP）和依赖项注入（DoI）简化构造业务 Bean 的，面向方面编程和依赖项注入是互补的关键技术。这样有助于高校评估 Web 应用程序中简化和纯化域模型和应用程序分层，其依赖项注入封装了资源和协调器发现的细节，而“方面”可以（在其他事情中）封装中间件服务调用的细节，如提供事务和安全性管理，同时方面可以帮助把“依赖项注入”的能力带到更广的对象和服务中，而依赖项注入可以用来对方面本身进行配置。而且是基于动态 AOP 机制实现的。为了实现 AOP 机制，Spring 默认情况下使用 Java Dynamic Proxy（JDP），但是 JDP 要求代理的对象必须实现一接口，该接口定义了准备代理的方法。而没有实现任何接口的功能，其 Spring 通过 CGLib 实现这一功能。否则用 Spring 框架提供的 BeanFactory 机制以及自定义工类来加载 Spring 的配置文件，在实现数据访问时，用 DAO 来实现数据访问，并将 DAO 注射到 Service 对象中，其调用这个 DAO 对象和与持久层通信，并通过 Spring 把 DAO 对象与 BSO 实现关联，同时进行 UDDI 中心注册供 BSO 发现以及 WSDL 描述相关的事务，供集成层进行异构集成。

（3）集成层。采用 Web 服务实现数据集成，是业务层中的一部分，其 Spring 也对其很好的支持，Web 服务标准技术是由 XML、SOAP、UDDI、WSDL 等标准协议和技术控制实现的，同时基于 Web Service 的系统都受到 WS-*系列约束和规范的，采用 Web Services 自描述特性和耦合性来发现和使用其他应用程序以完成任务，以及动态定位网络上其他组件并与之互动以提供服务，实现工具通过库提供的接口访问 UDDI 注册库以发现哪些服务可用，注册库中的所有注册项是公开的，也可以是私有的。在注册库中有可用服务的描述，其中 WSDL 格式的项描述了服务和接口，这个方式是静态查询，但应用程序是采用动态查询来满足自身的需求。图 2-40 与图 2-41 是 Web Service 与其他轻量级开源技术结构图与可扩展关系与交互图。

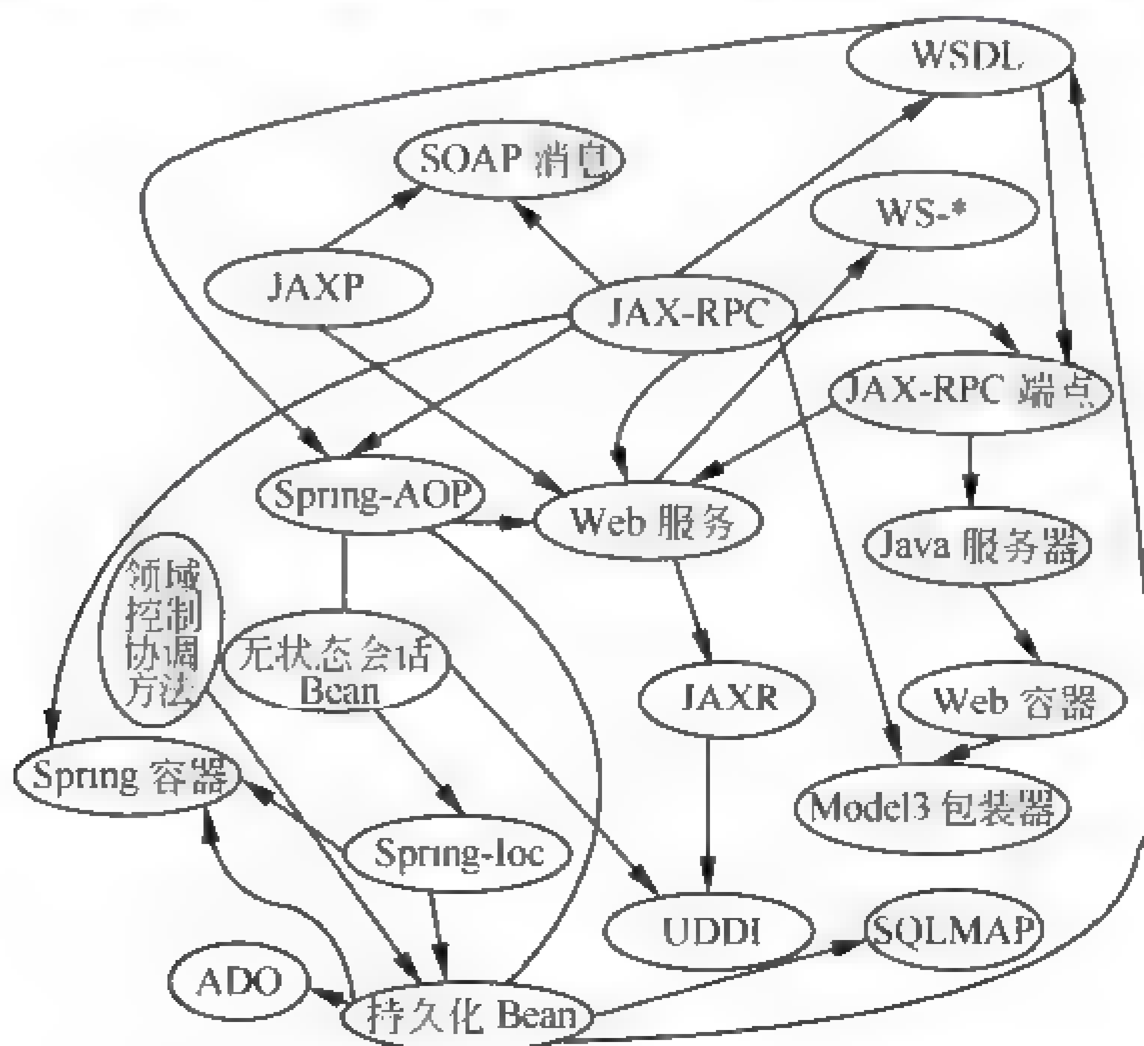

图 2-40　Web Service 与其他轻量级开源技术结构图

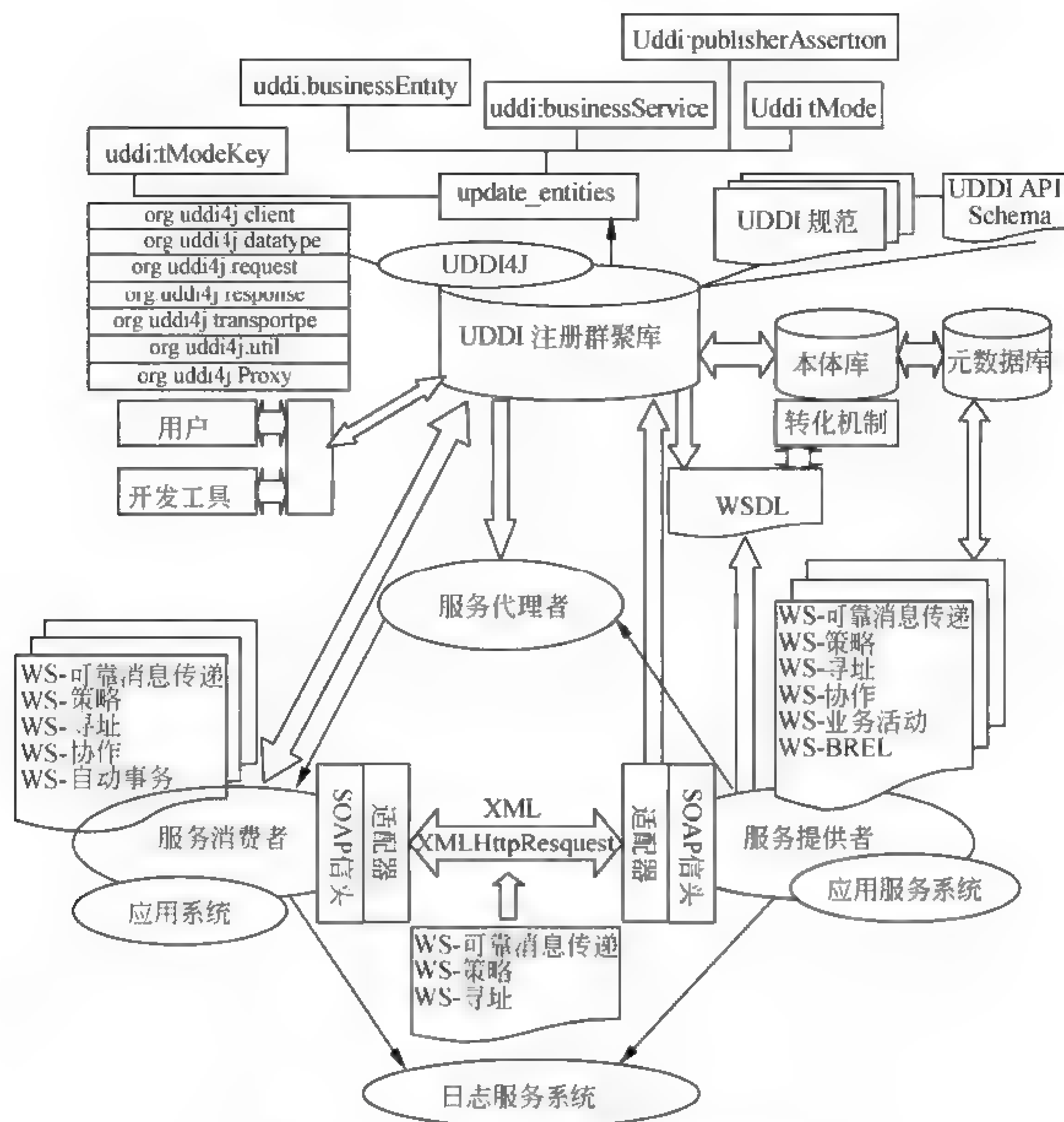

图 2-41　Web Service 可扩展关系与交互图

其中，通用、发现和集成（UDDI）定义了 Web Services 的发布与发现的方法且提供了一种基于分布式的商业注册中心的方法，其信息描述格式是基于通用的 XML 格式：

① 白页（White Page）：包括地址、合同和已知标识。

② 黄页（Yellow Page）：包括基于标准的行业目录。

③ 绿页（Green Page）：有关由企业公开的服务的技术信息，还包括对 Web Services 规范引用和 URL 的目录机制，其形式可能是一些指向文件或者是 URL 的指针。

④ 电子商务 XML（ebXML）：是有用以标准化电子商务数据交换。在企业间传送 XML 消息的传输可以是 SOAP 或者由 ebXML 提供的 SOAP 扩展，它的消息传输使消息可以安全和可靠的传输，不需要开发人员处理细节。

⑤ WS-*系列：是一种约束 Web 服务运行、集成、互用等规范和标准。

（4）域控制层。该层主要实现需求中的业务对象（BO）组成，主要关注来自域对象的数据，为它们公开相关的结构类型，如 JPetStore 允许将数据库中信息存放到域对象中，并提供了一个域类，而在 JpetStore 中用作数据传输对象（DTO），贯穿视图层、业务层和数据层，用于在不同层之间传输数据，这样即便连接断开的情况下也可以把数据显示到表示层，而那些对象同样可以返回给持久层，从而达到数据库里数据更新。在该层建立 XML 映射文件来管理持久性对象和保护值的对象，使之与数据逻辑和业务逻辑分离和增强数据访问的安全性，从而

提高逻辑访问的松散耦合度，顺利使 BO 在集成层中控制完成分布式数据交换和控制管理。

（5）持久层。采用 Hibernate 实现访问数据，Hibernate 是基于传统的 POJO 的开源代码持久性框架，它通过 XML 配置文件提供 POJO 到关系数据库表的与对象相关的映射（OR）。Hibernate 框架是应用程序调用的、用于数据持久性的数据访问抽象层。同时 Hibernate 还提供了从 Java 类到数据库表（以及从 Java 数据类型到 SQL 数据类型）的映射以及数据查询和检索功能。Hibernate 生成必需的 SQL 调用，还负责结果集处理和对象转换。在实际操作中，可以使用 Xdoclet 工具（详见 6.7.2 小节）来生成 Hibernate 的 XML 映射文件通过创建的 DAO 接口交换数据。在实现接口操作时，通常首先得到 Hibernate 的 session 对象实现对数据对象的相关操作。而 Spring 框架对 Hibernate 良好的支持，在事务管理、session 管理等方面进行了封装，然后通过配置 hibernate.cfg.xml 实现 Spring 的 Bean 与 Hibernate 通信。

图 2-42 是整个轻量级的开源技术整合的时序运行图，该图表达了各层通信情况与各开源技术间的整合。

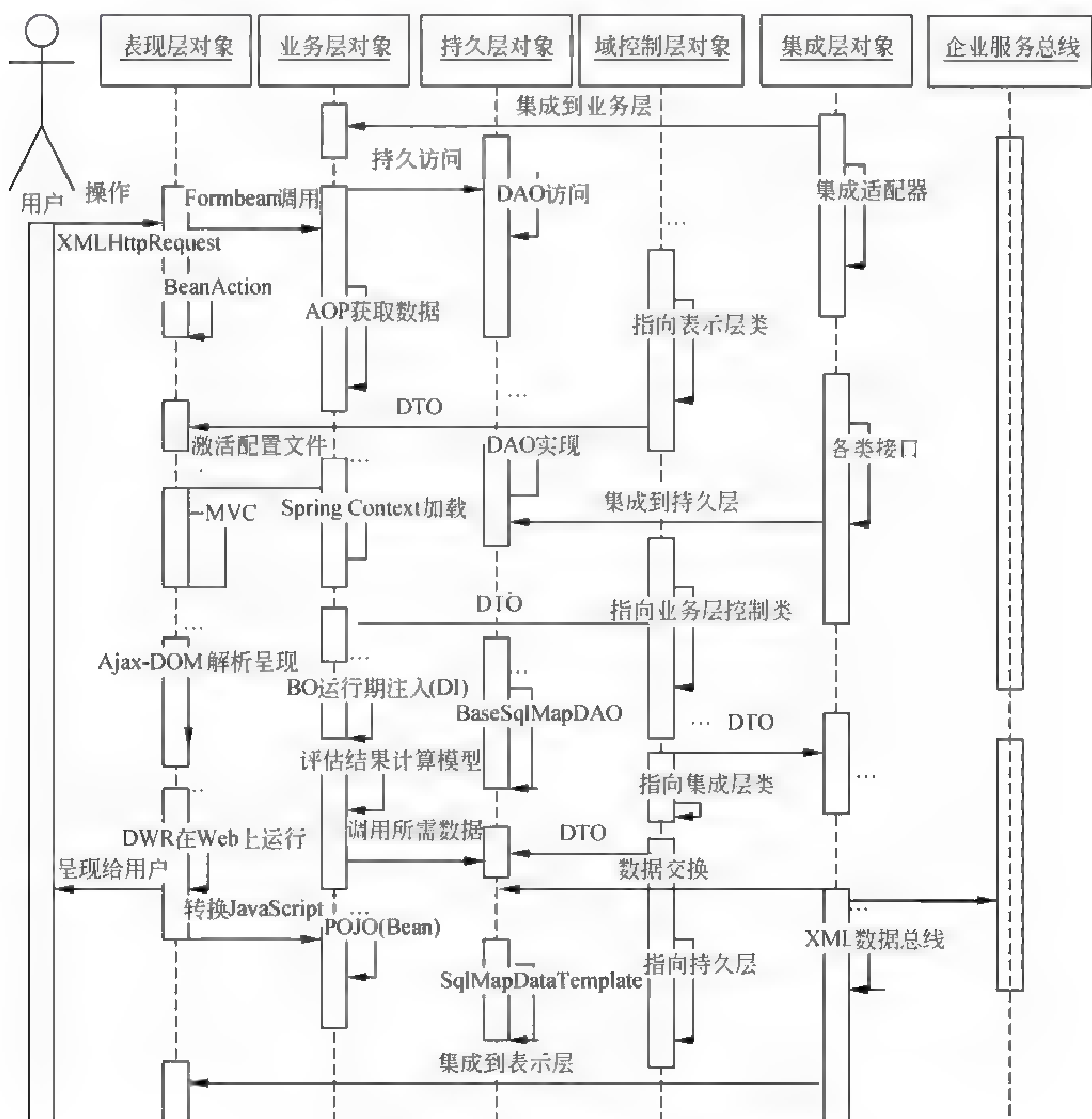

图 2-42 面向轻量级的开源技术整合的时序运行图

图 2-43 所示是 SSi 轻量级软件层次结构。

SSi 与 SSH 不同在于持久层处理上。持久性是从永久性存储器中读取对象、将对象写入永久性存储器和从中删除对象的能力，是指对象的生存特性。如果对象的生存期跨越程序执行期，则称此对象具有持久化性质；持久化访问是从系统类通过 SQL 直接访问关系数据，再从数据类通过 SQL 访问关系数据，然后发展到目前通过专门的持久层来访问关系数据的过程，图 2-44 是持久层存储结构图。持久层将 Java 对象映射到关系数据库表来获得持久性，而持久性框架是通过相关的数据类从数据库载入一系列的对象，并进行操作存储到数据库中，来满足持久化要求。目前有以下几种持久化方法：JDBC 框架、EJB、Hibernate、JDO、ORM 以及本书中研究的 iBATIS，如表 2-16 所示。

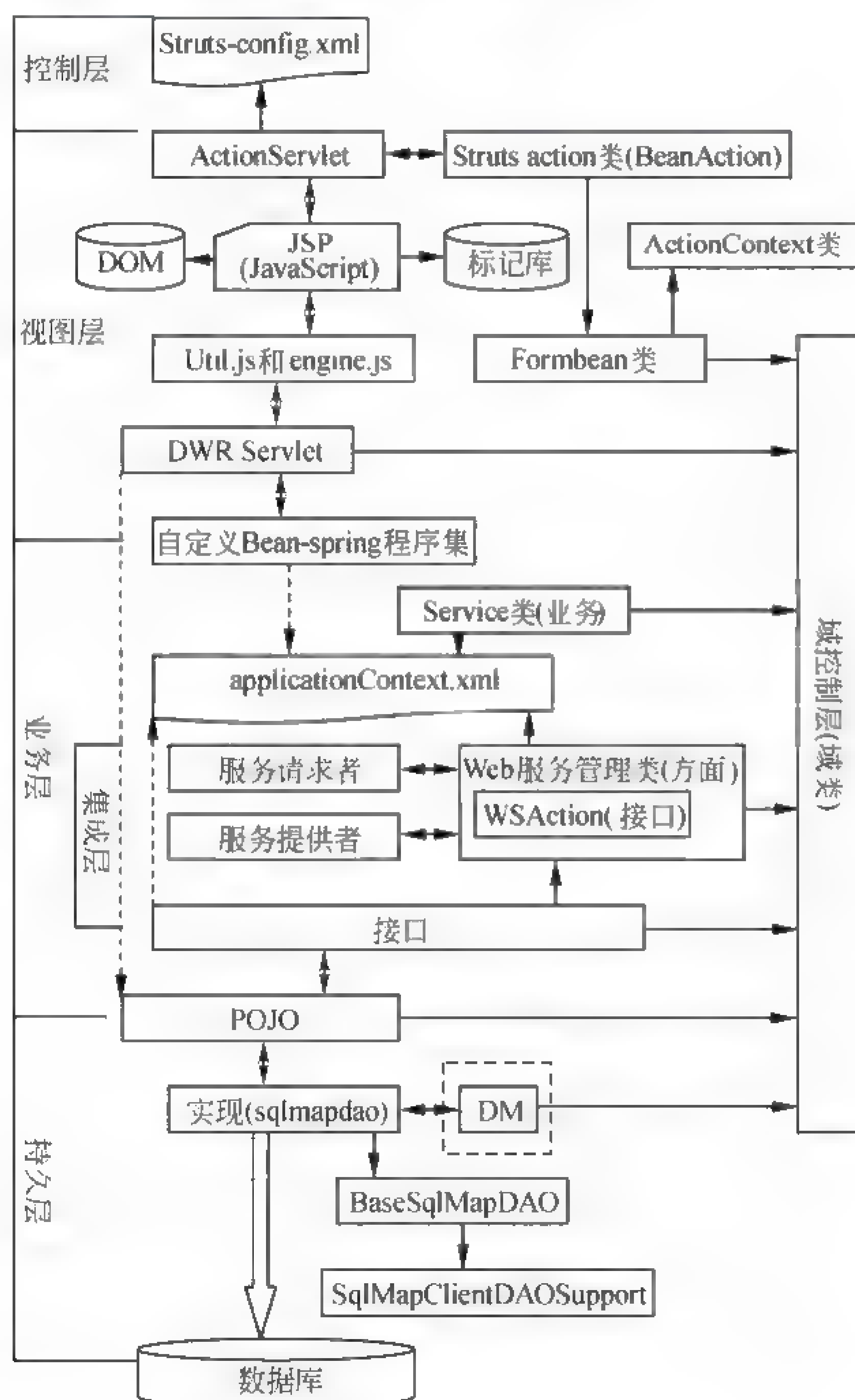

图 2-43　面向轻量级的 SSi 层次结构图

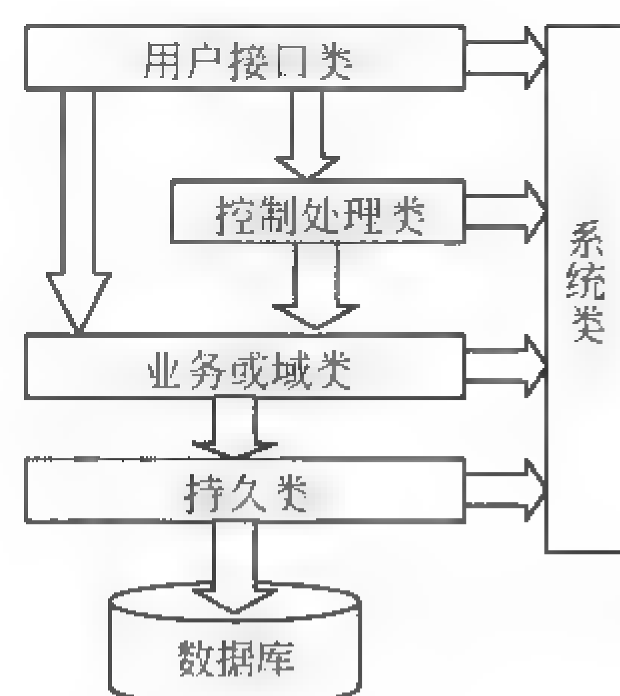

图 2-44　持久层存储结构图

该 iBATIS 原则为：

① SQL Maps。Data Maps 框架可以专门用于 OR 映射，OR 映射是 Java 域对象到数据库中关系表的映射，它是使用 XML 描述符将 JavaBeans 映射到 SQL 语言，是执行 SQL 并将

结果映射回对象的框架。且有助于极大地减少访问关系数据库通常所需的 Java 代码量。其核心功能是围绕 Mapped Statement 进行的。Mapped Statement 可以拥有 Parameter Map 和 Result Map 框架。因此 Mapped Statement 实质上是一个 XML 元素，该元素包含负责执行某些操作并将输入/输出参数映射到 Java 对象的 SQL 语句；其 Parameter Map 为 Mapped Statement 提供数据输入参数，而 Result Map 类似于 Parameter Map，主要用于数据输出，且 Result Map 允许将 Mapped Statements 映射回 Java 对象的方式。框架中的 TransactionManager 元素允许在给定配置的情况下按需求对事务服务进行配置。在实现 OR 时，通常是通过配置 sql-map-config.xml 来定义与事务管理有关的项，以及如何连接到数据库。同时，指定包含 Mapped Statements、Result Map 等事项的*.xml 文件列表也是在这里进行的。然后通过 Maps API 将 JavaBeans 对象映射到 PreparedStatement 参数和 ResultSets，并完成持久化操作。图 2-43 是 Data Maps 框架的结构图。

表 2-16　各持久化结构比较表

特征 / 名称	特点	优点	缺点
JDBC 框架	是一种基本的 Java 技术 API，允许直接访问数据库，它代表最低级别的持久性策略，通过把数据访问包装在数据访问对象中，而对象中包装有一个关系表	良好的控制能力、访问数据库的所有权限、灵活性	经常是 OOP 对象、性能、持久性弱、操作复杂
EJB	为企业级 JavaBean 提供两种标准化的持久性策略，EJB 1.x 标准 和 2.x 标准，一般不使用这种持久访问策略	有时便于操作 JavaBean	操作复杂、持久性弱
Hibernate	目前，可说是和一种持久性的标准，它快速，有效，而且是免费的，可任意制定的 POJO 持久性，其通过反射机制来实现透明度，以及通过动态代理的方式实现数据持久操作。特别地它仅仅只支持关系数据库	灵活的映射机制、较强的持久性，与其他轻量级开源软件集成、兼容、快速和有效	难管理、框架性不够强，目前只支持关系数据库较强
JDO	提供一种叫做 JDO QL 的查询语言，且为任意的数据存储提供透明的持久性。其 JDO 通过分离式处理（detached processing）提供一个模型，以使能够从一个 JDO 会话（叫做 Persistence Manager）分离一个对象，改变并重新附加该对象，然后将数据库的应用进行更改	执行速度快、管理性较强，根据映射支持用户界面，具有较好的映射机制，持久性一般	使用的标准不成功、市场前景弱
ORM	主要用于关系模式或一个可能频繁改变的关系模式，且进行相关的管理配置和依赖性的策略，此时，需要选择一个事务策略并使用数据源和连接池	提供一种 OOP 与关系数据库的映射机制、有较好的持久性	操作方便且复杂
iBATIS	iBATIS 是一个对象关系（object-to-relational）映射数据层和 Java™ 的框架，由 SQL Maps 和 Data Access Objects（DAO）两部分组成，且它们之间可以结合使用，也可以单独使用	用 XML 指定查询对象的映射、独立于代码库的 SQL 和限定的关系管理、不必担心对象/关系的不匹配，可伸缩性好、技术性好	太依赖于 SQL

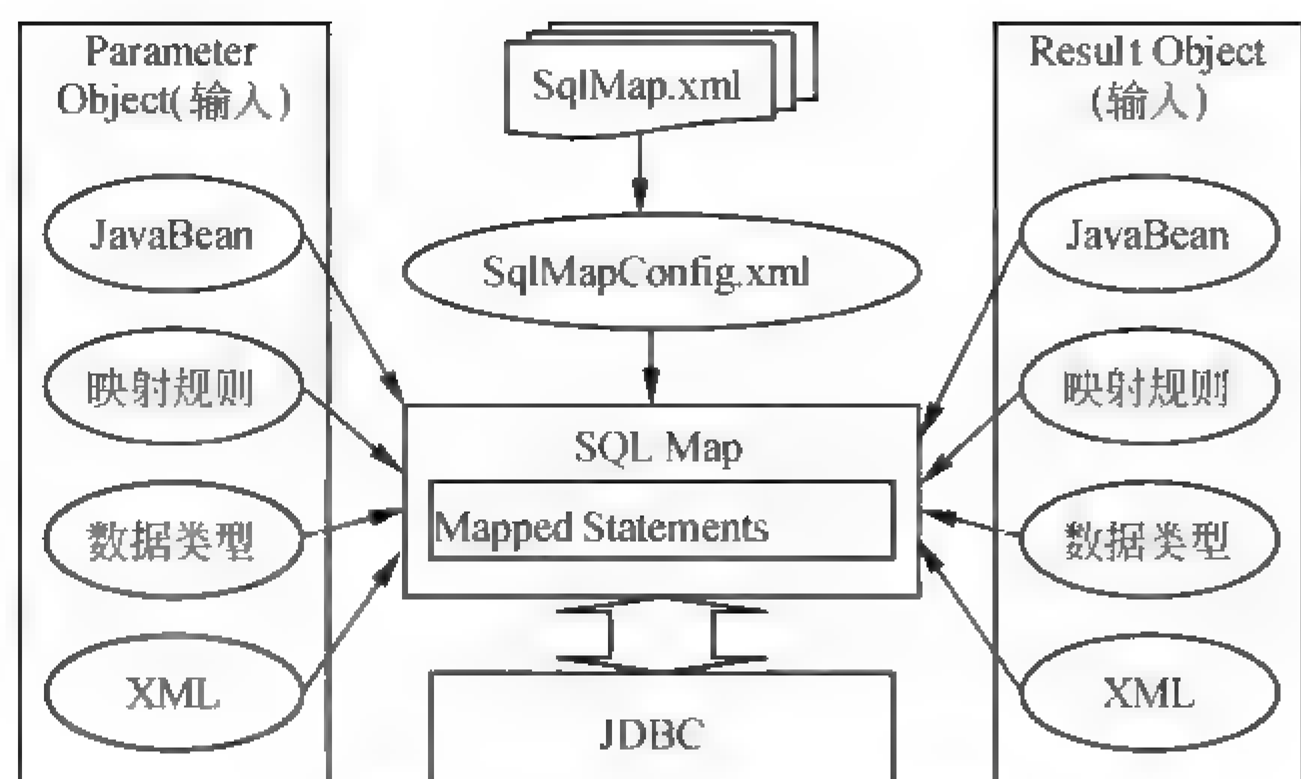

图 2-45　Data Maps 框架的结构图

② Data Access Objects。DAO 框架的主要目标是抽象化应用程序的数据访问层和持久层的表示方式和位置，使它远离应用程序的业务逻辑。同时 DAO 框架允许在应用程序中定义负责数据中心操作的接口，使用 DAO 可以动态地配置应用程序，从而使用不同的持久性机制和隐藏持久性层实现的细节。如果如果应用程序通过 JDBC 来获取持久性，则 DAO 框架的目标就是抽象 Connection、PreparedStatement 和 ResultSet 等类和接口的使用，并下移到持久层。如果应用程序使用 HTTP GET 和 POST 来获得和存储数据，则 DAO 框架的用途是实现抽象化类 HttpUrlConnection 等的使用，使它们远离应用程序的业务层。然后应用程序可以使用 DAO 接口在数据上执行操作，这样就可以从数据库、Web 服务或其他任何源中获得数据。图 2-46 是 DAO 结构图。

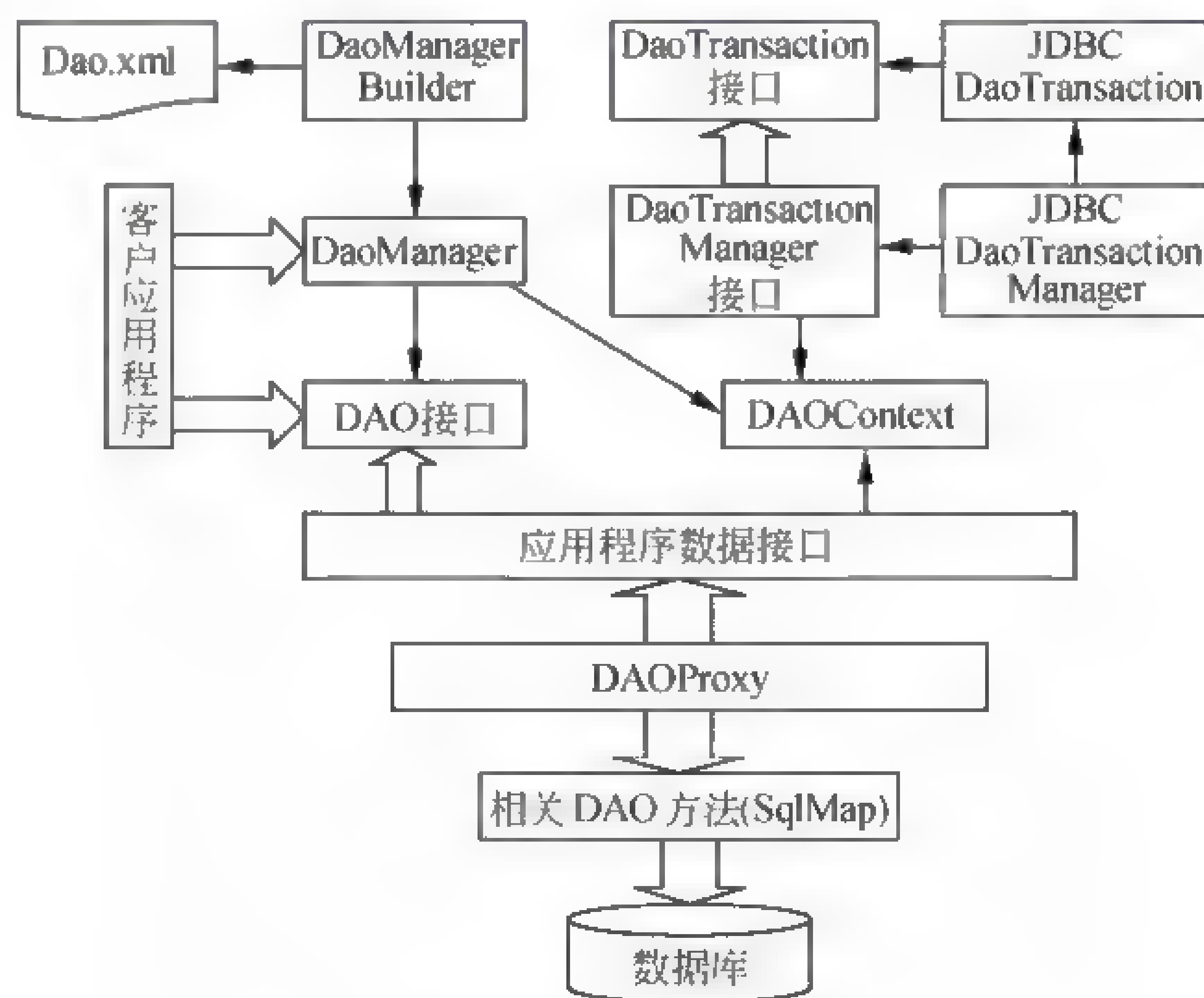

图 2-46　DAO 结构图

③ 一个持久框架。JPetStore 相关技术。JPetStore 是基于 Struts 开发的，是一个结合 SQL Maps 和 DAO 框架间协作的典型程序，它适用多种数据库。它的数据访问方式也是高度模块化且相当实用的，因它 JPetStore 是实现数据持久化一个很好模型，很多功能在开发时可以直接使用。其中有一个 CatalogService 类负责维护 DaoManager 类和 Dao 接口的引用，并把相

关的数据操作公开给 JpetStore，最终完成 OR 映射。具体的说：表现层框架使用 Struts 实现。持久层使用 SQL Maps 框架和 iBATIS DAO 框架组成。应用程序使用 Derby 数据库进行永久存储。而本书还重点研究业务层有 Spring 实现，同时对表示层还引入 Struts 更多的功能。而 Derby 是一个与平台无关的数据库引擎，它以 Java 类库的形式对外提供服务。由于 Derby 是使用 Java 语言编写的，且支持 JDBC 和 SQL92E 标准，因此可以将其嵌入到 Java 程序中并可以在各种大型关系数据库移植，以及嵌入到 Geronimo J2EE 服务器中，并将其作为一个单独的包进行部署。而 Geronimo 是一个完全兼容的大型开放源码 J2EE 应用服务器，目前，它集成了 Apache 下相关的所有开源件，包括基于 JSP、servlet、EJB 的 Web 应用程序，关系数据库服务，基于 JMS 的消息队列组件，MDB 组件，基于 Spring 的轻量级应用程序组件，Web 服务，JBI 服务组件等。图 2-47 所示是 JpetStore 运行框架。

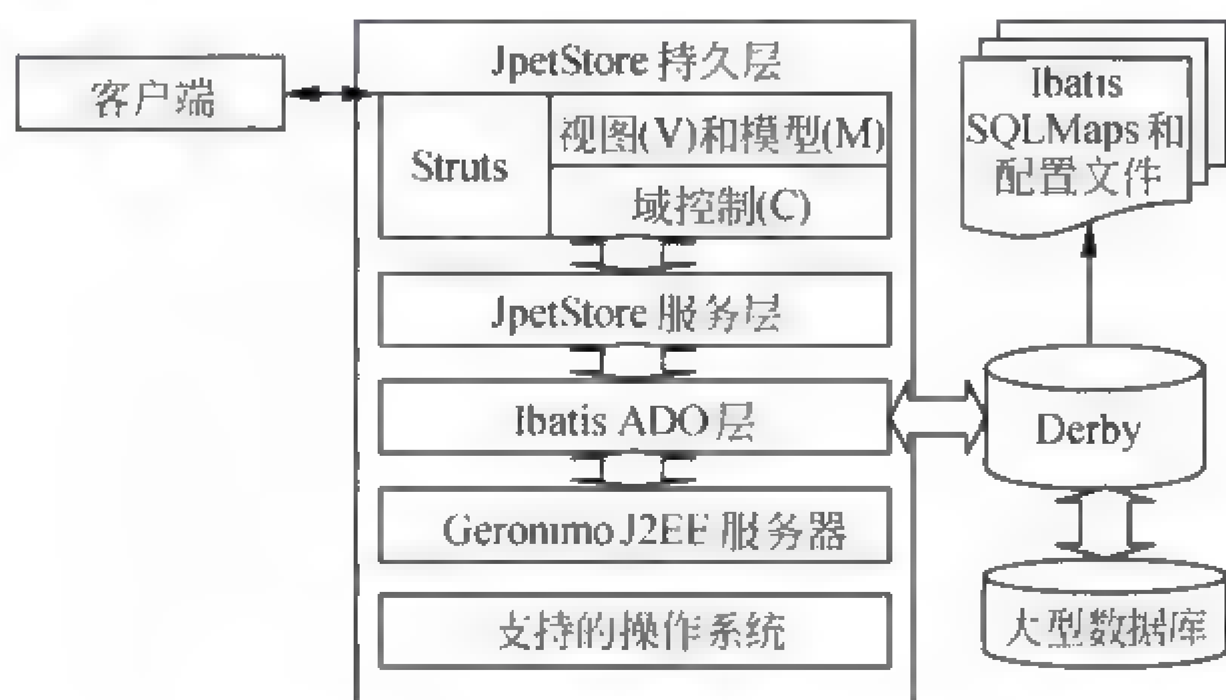

图 2-47　JpetStore 运行框架

2.3　可信软件的构架方法

面向可信的开源软件构架方法是根据可信的原理、方法和技术来指导面向开源软件的软件开发，这样可以为多源、多样本的开源软件提供一种可信的选择、研发和应用的方法。下面主要从可信软件概述、可信软件原理、可信软件构造、可信软件演化、可信软件度量、可信软件技术等角度分析可信软件的基本方法来为面向开源软件的软件研发提供参考和应用的指导。

2.3.1　可信软件概述

通常情况下，可信软件是保证软件运行时的稳定性、可靠性、安全性、正确性和生存性等软件属性的关系。当前可信软件是软件研究领域中重要的研究内容，也是促进软件可持续、高质量发展的重要的基础研究方向。国家自然科学基金、国国家高技术研究发展计划（863）、国家重点基础研究发展计划（973）连续发布了可信软件基础研究课题来增强可信软件的基础性的工作，从而为可信软件的有效应用提供理论支撑。而可信计算（Trusted Computing，TC），是一项由可信计算组（Trusted Computing Group）推动和开发的技术，这个术语来源于可信

系统（Trusted system）；TCG 将可信计算定义为如果一个实体的行为总是按照预期的方式和目标进行，那它就是可信的，并且还有其特定含义。这项技术的支持者称它将会使计算机更加安全、更加不易被病毒和恶意软件侵害，因此从最终用户角度来看也更加可靠。此外，他们还宣称可信计算将会使计算机和服务器提供比现有更强的计算机安全性。而反对者认为可信计算背后的那些公司并不那么值得信任，这项技术给系统和软件设计者过多的权利和控制；他们还认为可信计算会潜在地迫使用户的在线交互过程失去匿名性，并强制推行一些不必要的技术。同时 Ingrid Marson①、Bruce Schneier②、Bruce Schneier（著名密码学家）等专家也反对可信计算，因为他们相信它将给计算机制造商和软件作者更多限制用户使用自己的计算机的能力。但不管这场争论以及可信计算最终产品的形式怎样，在计算机领域拥有重大影响的公司，如芯片制造商 Intel 和 AMD 和 Microsoft 系统软件开发商和 GNU，都计划在下一代的产品中引入可信计算技术，如：Windows Vista、GNU GPLv3。

尽管赞成者声称可信计算增进了安全性，而批评者则反击称不仅安全得不到增强，可信计算还可能推动强制数字权限管理（DRM），伤害到用户的隐私，并对用户强加其他的限制。而支持者又争辩称：隐私问题在现有的规范中已经解决——可能是对早期版本规范受到的批评而做的相应处理。

然而，一个完全可信的系统是主要由认证密钥、安全输入输出、内存屏蔽/受保护执行、封装存储、远程证明等五部分构成。并且反对者也指出一个可信系统也存在这些问题：用户不能更换软件、用户不能控制他们接收的信息、丧失互联网上的匿名性、属主重载（owner override）等问题。

当前的可信软件不仅要研究可信计算中的安全问题，更是要研究软件整个过程中（生命周期）的缺陷，使软件是在正确性、可靠性、安全性、时效性、完整性、可用性、可预测性、生存性、可控性等特性等上的众多概念基础上发展起来的一个新概念，是客观对象诸多属性在人们心目中的一个综合反映。因此，一般认为，“可信”是指一个实体在实现既定目标的过程中，行为及结果可以预期，它强调目标与实现相符，强调行为和结果的可预测性和可控制性。软件的“可信”是指软件系统的动态行为及其结果总是符合人们的预期，在受到干扰时仍能提供连续的服务。这里的“干扰”包括操作错误、环境影响、外部攻击等[30]。

随着的软件规模的不断扩大，其内部结构越来越复杂，应用环境越来越开放，这些因素使得人们更加关注于软件的可信性问题。与可信计算一样，目前关于可信软件，学术界也有着不同的理论和认识。

国内外由于软件缺陷而导致的重大灾难、事故和严重损失屡见不鲜，造成的经济损失也是非常巨大的，2002 年 6 月 28 日美国标准技术研究所（NIST）就估算，美国每年因软件失效所造成的年度经济损失近 600 亿美元，约占其 GDP 的 0.6%。如何构造出缺陷较少的软件并保障其安全可靠运行，已经成为软件研发者义不容辞的责任。表 2-17 所示是近 20 年主要因软件缺陷而造成的重大事件和因安全问题造成的相关事件：

① http://www.zdnet.co.uk/news/security-management/2006/01/27/trusted-computing-comes-under-attack-39249368

② http://www.schneier.com/crypto-gram-0208.html

表 2-17　因软件缺陷而造成重大损失的事件

序号	年度	事件名称	原因	经济损失
1	1994 年 12 月 3 日	美国弗吉尼亚州 Lynchburg 大学的 T.R.Nicely 博士使用装有 Pentium 芯片的计算机计算时发现错误	Pentium 芯片刻录了一个软件缺陷（浮点除法）	4 亿美元
2	1996 年 6 月 4 日	欧洲阿丽亚娜 5 型火箭的首次发射	惯性参考系统软件的数据转换错误引起操作失误	25 亿美元
3	2003 年 5 月	俄罗斯“联盟-TMA1”载人飞船在返回途中偏离预定降落地点约 460km	飞船的导航计算机软件设计错误	不详
4	2003 年 8 月 14 日	美国及加拿大部分地区历史上最大的停电事故	俄亥俄州的第一能源公司下属的电力监测与控制管理系统 XA/21 出现错误，系统中重要的预警部分出现严重故障，负责预警服务的主服务器与备份服务器连接失控，错误没有得到及时通报和处理，最终多个重要设备出版故障	60 亿美元
5	2004 年 9 月 14 日	美国洛杉矶机场 400 余架飞机与机场指挥系统一度失去联系	空管软件中的时钟管理缺陷	不详
6	2005 年 4 月 20 日	中国银联系统通信网络与主机出现故障，造成辖内跨行交易全部中断	银联新近准备上线的某外国设备的隐性缺陷诱发了跨行交易系统主机的缺陷，使主机发生故障	不详
7	2005 年 11 月 1 日	日本东京证券交易所股市停摆	由于软件升级出现系统故障	不详
8	2006 年 3 月 2 日	沪深大盘忽然发生罕见大跳水，七分钟之内上证指数跌去近 20 点	当日下午刚上市的招商银行认股权证成交量巨大，导致其行情显示总成量字段溢出，使价格在股票分析软件上成了一条不再波动的直线	不详
9	2006 年	中航信离港系统就发生了三次软件系统故障	造成近百个机场登机系统瘫痪	不详
10	2007 年 8 月 14 日	美国洛杉机国际机场计算机发生故障，60 个航班的 2 万旅客无法入关	包含旅客姓名和犯罪记录的部分数据系统瘫痪	不详
11	2007 年	“熊猫烧香”病毒	一夜之间就使上百万台计算机感染并遭到损害	不详
12	2006 年 4 月 20 日	中国银联跨行交易系统出现故障	整个交易系统瘫痪约八小时	不详

不仅出现表 2-17 所示与可信相关的事件外，近年来，国内 2006 年我国共查处境内信息网络犯罪案件 41379 件，比 2005 年增加了 98.9%；并且境外敌对势力也利用软件缺陷对大陆实施信息攻击计划（例如木马计划、达尔文计划、茉莉计划），严重危害着国家安全；等等。这些不可信事件的出现，使得软件可信性问题已经成为国际上一个普遍关注的问题，因此如何构造出缺陷较少的软件并保障其安全可靠运行，已经成为软件研发者义不容辞的责任，也是保障国家、个人安全必须研究和实行的问题。同时，在运用开源软件构架软件系统时也不

得不考虑可信性在开源软件的地位和作用，使其所构建的软件可信、可靠。

基于此和国家自然科学基金委员会在广泛听取专家意见和反复深入研讨的基础上，国家启动了可信软件基础研究计划来推动软件基础理论的探索与创新，来应对软件发展的重要科学挑战，以促进我国软件产业的发展[30]。

2.3.2　可信软件基本原理

可信软件基本原理就是在满足软件正确性、可靠性、安全性、时效性、完整性、可用性、可预测性、生存性、可控性等原则构建的软件系统。即可信软件通常是指在特定环境下其运行行为及其结果符合人们预期，并在受到干扰时仍能提供连续服务的软件。

软件可信性需要从软件内部质量需求的观点和外部质量需求的观点来认识。软件可信性与软件行为、应用需求有紧密关系，在软件开发的不同阶段，随着软件的不同，呈现关注点不同的各种属性形成。软件的发展、应用的发展赋予了软件可信不断发展的内涵。即随着软件形态及其作用的发展，可信的概念是在不断丰富和发展的。准确把握软件可信性生命周期要对可信有个准确的表达，而它作为属性，在表达时需要承载者，即软件，不同的可信需求对软件生命周期的不同阶段提出了不同的内部质量需求，相应地也提出不同的可信性侧面属性要求[31]。下面就根据参考文献[31]从软件可信性建模、软件可信性设计、软件可信性验证、软件可信性演化、软件可信性风险控制等五个方面概述可信软件的原理。

1．软件可信性建模

根据可信性定义可知，确立高可信的基础就是为软件可信性进行建模，而可信性建模的关键是需要对期望系统行为进行恰当的描述，或者是逼近，或者抽象。但事实上程序的行为是由许多细节功能、需求来形成软件可信性和复杂性的。因此，如何准确地表达期望是可信软件建模的关键，它包括了环境、系统、环境与系统交互的可信性。从而使得这种表示可信期望的建模涉及来自各方的约束和影响，特别是受到系统本身的潜在的缺陷、安全性、可靠性等影响，这就对可信软件建模提出了更高的要求和难度。截止目前，对可信软件的建模还未形成一套统一的理论、方法来指导可信软件的建模。

2．软件可信性设计

和其他软件一样，软件可信性是设计出来的，从对可信建模可以看到，高可信软件的设计必须放在一个环境和系统中来认识，再根据软件本身潜在缺陷、安全性和可靠性来进一步认识。软件可信性问题仍是一个系统的问题，是一个以系统思想为指导的软件构架的软件体系结构。因此，从软件体系结构的角度提高系统的可信性是一个必然选择，也是工业界和学术界一个热点问题，人们研究了诸多注重提高可信性的体系结构，如体现冗余性、容错性、实时性及资源约束性等，将可信性贯穿于设计的各个环节。让体系结构不仅应支持功能和非功能需求，而且也让支持对环境的适应性和在环境支持下的动态演化。然而怎样从设计上建立软件运行时对高可信性质的动态保证是近来发展的重要技术，其代表是软件自诊断和恢复，并且近年来，一些面向应用相关策略的软件运行时验证技术发展起来，它成为了软件容错设计的重要策略。其思路是面向用户需求综合设计出监控程序，对程序行为通过探斟来获取运行时的监控。同时，近年来软件实现的硬件容错技术为抗恶劣环境计算机系统的可信提供了低成本、高性能的途径，这为软件可信性提供了环境的支撑，也为可信软件的设计带的了环

境和安全的支持。

3．软件可信性验证

在定义了软件可信性期望的行为后，一个自然的想法是拿到软件系统的行为并与期望的行为对比。因此，捕获可信性的关键是如何获得系统的行为。事实上，测试、验证、运行时监控等都是对系统行为的认识。然而要获得系统的行为的证据，需要论证证据对可信性的支持。这实际上是论证获得证据的方法的科学性。因而，形式化方法在可信软件技术中有重要的应用与实践，即用形式化来抽象系统证据的行为，然后从语义角度进行对系统的分析验证。但从技术角度看，软件分析以软件为对象通过静态或动态的方法进行人工或自动分析，以验证、确认、监控或发现软件性质，这些性质包括规约和约束。因此，相继出现了多种以形式化和抽象化为基础的软件分析验证工具，其中以法国巴黎高等师范的基础于抽象解释的软件分析工作 ASTREE 最为著名，这是因为这个工具成功地分析了空中客车 340/380 上的飞控软件无运行时错误。而且近来，该工具又应用于空间货运飞船的对接软件。但由于抽象语义基础不同，其方法的可靠性和精确性也不同，抽象的复杂性导致其开销和代价也不同。

模型检验也是形式验证系统的另一个令人瞩目的途径，它的基本思想是把待验证的系统抽象为有穷状态系统，并使所期望的系统性质用时序逻辑公式等来表达。而模型检验技术通过有穷状态上的高效搜索来决断待验证系统是否满足所期望的性质。如果不满足，则给出一个系统为反例说明系统为何不满足性质。

由于捕获可信性实际上是对系统行为一种逼近，使得不同的近似形式的证据反映了可信性的不同侧面。因此，也就有了不同方法和结果的互补性；人们往往会根据应用的需要，从不同的选择来获得证明，如代价、精确性和扩展性等。一般来说，测试和静态分析的扩展性好于模型检验，但是精确性则模型更好，运行时监控的扩展性也是模型好。同时，由于动态的方法，在实时应用中有开销。这时，把静态与动态分析验证技术结合起来是未来可信性保证的重要方法。当在可信性评估时也强调证据手段的多样性，从而建立一个综合的证据链。而评估的结论可以综合运用测试方式获得的证据与通过验证方式获得的证据。

4．软件可信性演化

为了获得满足需求的软件可信性，形式化方法和工程化方法取得了很好的效益，在此小节进行了软件可信性的概述与分析。而形式化方法核心思想是通过严格的数学化方法建立面向软件可信性的形式系统，给出用户需求的形式化规约，在此基础上，形式地证明目标软件符合规约，或通过形式变换由规约产生符合规约的目标软件，软件的可信性将通过系统的可靠性保证。形式化方法的实践产生了一系列有价值的软件可信属性的验证方法、系统和工具。工程化方法的核心思想是建立严格的工程规范，在软件生命期的各个阶段，通过规范管理和辅助工作，最大限度地减少人为错误机会，或尽可能早地发现人为错误。实践表明，工程化方法是开发可信软件的有效途径。

目前更为突出的是，特别是进入网络时代，软件在开放环境下呈现了演化特性，随之而来，软件的可信性在环境变化下也会动态变化，软件的可信性需求也会变化。如何控制软件的可信性的演化成为重要的研究内容和方法，使其在网络下演化为可信的软件。目前，软件演化的形式大体可以为三类：第一是开源软件的版本演化，第二是基础软件的在线升级，第三是服务软件的在线维护。特别是网络环境下的软件开发演化与运行演化已呈现出交织协同、并发共长的态势，单纯的开发演化或者运行演化已经不能适应快速演化的需要。软件运行状

态改变的同时软件版本也不断升级，从而形成了网络时代演化特征，即可信软件可以在一个社会化的网络环境中演化而来。同时，网络环境下的软件演化具有很多新特点，包括演化数据的分散性与多样性、演化资源的异构性与泛在性、演化过程的交叠化、演化目标的多元化等。这要求在实现软件可信性演化控制上需要遵循支持持续演化支持机制引导、支持在线演化等原则进行可信软件演化管理。

5．软件可信性风险控制

可信软件的风险是存在的，而且是伴随着整个可信软件的生命周期。在参考文献[32]中提出了一种人因—技术—组织—环境—过程的多维约束满足风险控制的模型。该模型实现的核心思想是：将软件可信性描述为一些需要达到的约束要求，当软件开发中所有条件符合所定义的约束要求时，则称软件质量达到可信目标，其求解过程称为可信软件质量约束满足过程。而在风险控制过程中，首先，通过分析软件开发信息交流机制、软件开发模型及开发技术、工具和环境，定义面向可信软件的风险控制约束条件；而后，根据软件可信性要求，评判多维约束条件是否能够满足，进行控制模型的动态修正反馈，挖掘开发中的风险控制约束条件，同时对风险因素引起的软件信息失真、信息缺失、技术冲突等问题进行协调处理。当定义的约束条件都满足时，意味着软件开发阶段风险控制目标的完成，从而实现风险的计划、跟踪与处理。图 2-48 所示是一种基于多维约束满足的风险控制模型。

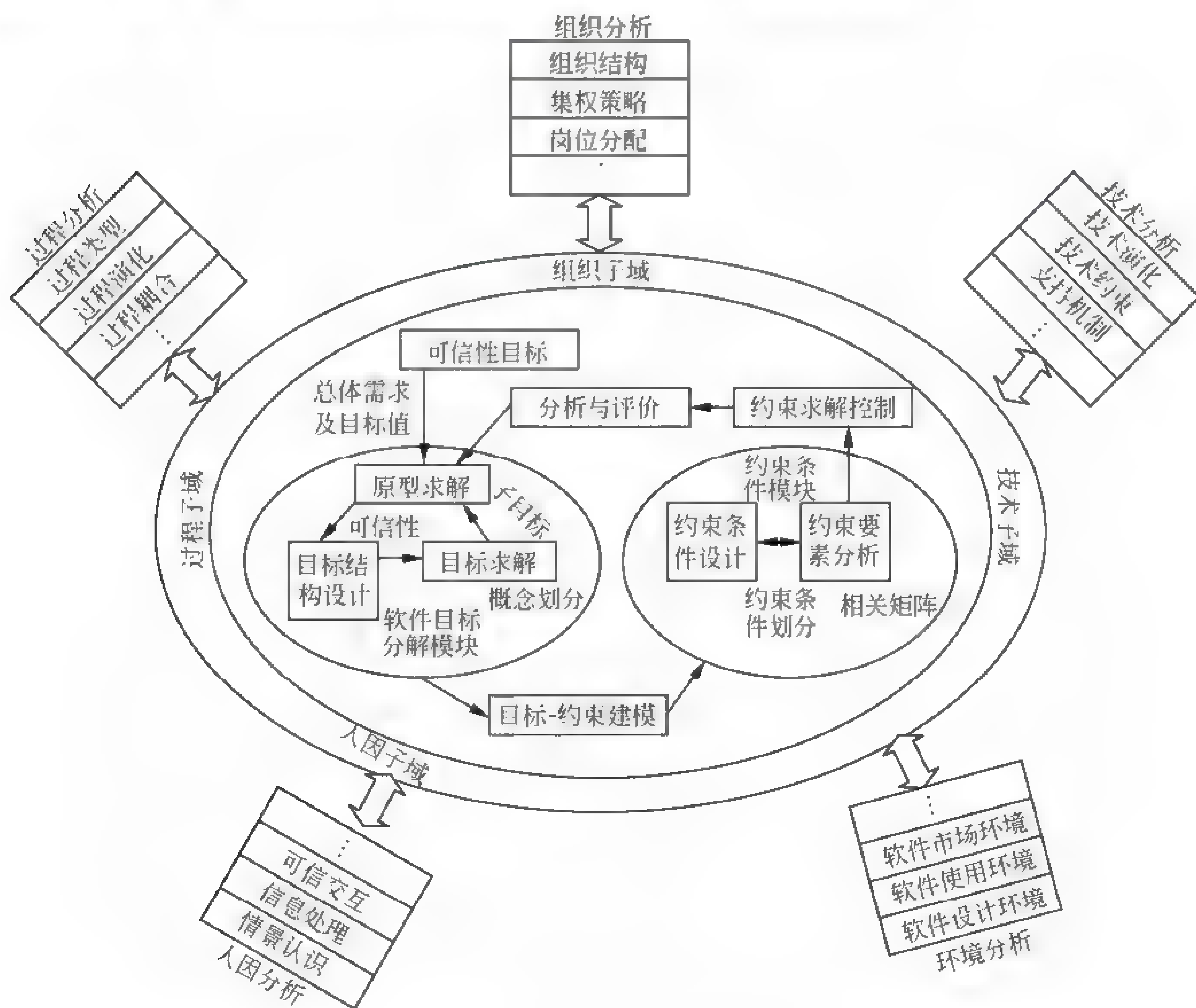

图 2-48　基于约束满足的可信软件风险控制模型[32]

2.3.3 可信软件构造所满足的基本条件

可信软件的构建是在可信原理支持下进行的，而可信软件构造就是在满足安全性、可靠性、正确性等条件下按需求构建软件的过程。而可信软件构造最终目标是揭示软件可信和环境可信的失效、度量和演化的基本规律，建立可信软件及其环境的构造与验证、演化与控制的方法和关键技术体系，研究可信软件开发工具和运行支撑平台及环境，并在典型的嵌入式软件和网络应用软件中进行验证和示范，促进软件从传统的单一度量理论到综合性的可信性度量理论及其构造方法的集成升华[30]。下面根据参考文献[30]从可信软件的度量、建模与预测，可信软件的构造与验证，可信软件的演化与控制，以及可信环境的构造与评估进一步概述可信软件在可信软件原理下的构造所满足的基本条件。

1．可信软件的度量、建模与预测

传统的软件理论是围绕程序正确性建立的，对正确性的刻画以定性方法为主，并且是以静态确定性的表达给出。对于可信软件，需要考察包括正确性、可靠性、安全性等诸多属性的综合度量空间，形成对软件可信性的科学理解，以定量方式给出可信性建模的系统方法论，以及适应环境依存稳定性条件下的可信度动态演化特征。因此，必须从如何认知软件可信性的角度建立新的软件系统方法论，从如何表述软件可信性的角度建立可信需求的建模、规约和分析方法，从如何把握软件可信性的角度揭示软件可信性演化的基本规律，从而解决软件系统可信度量标准和如何在其工作环境中进行评估的问题，对软件系统的可信性建立分级，并提供量化指标。主要研究内容包括：软件可信性度量、软件可信性的演化与预测、可信软件的风险及过程管理等。

2．可信软件的构造与验证

传统的软件理论在软件构造与验证时只注重在封闭环境下追求不可演化的绝对正确和效率优先。对于可信软件，必须适应开放环境下物理世界中的计算规律，从追求软件绝对正确和效率优先的软件方法学变为力求保证可演化的软件可信性满足需求的软件方法学。因此，如何进行可信性算法设计和软件设计、如何消解多属性引起的可信性冲突、如何进行可信性保证是解决可信软件开发问题的关键。主要研究内容包括：可信软件的程序理论与方法学、可信软件的需求工程、可信软件设计、构造与编译和可信软件的验证与测试。其中可信软件的需求工程是研究面向可信性的需求分析方法，研究基于社会的可信模型的需求工程方法，研究软件可信性的性质获取与形式规约方法，研究多维异质非功能需求的冲突消解与完整性表述方式方法，以及探索基于领域知识的可信性分析方法和理论。

3．可信软件的演化与控制

传统的软件理论仅从静态的角度认识软件部署后的变化，对于软件的维护往往是事后的被动响应，而开放环境下软件的演化是软件面向可生存性需求的重要特征。对于可信软件，需要从事后维护向事前设计、主动监控变化，形成对软件动态演化中的可信性控制方法。因此，如何认识环境的演化和软件自身的演化、如何动态获取可信性和控制可信性的变化、如何构建可信的运行平台是解决可信软件在开放动态环境中可信运行问题的关键。主要研究内容包括：可信软件运行监控机理、软件可信性动态控制方法。

4．可信环境的构造与评估

软件可信性离不开环境的支撑，环境可信是可信软件的重要方面。但是可信环境的基础

理论研究滞后于可信计算技术的研究，需要探求在网络环境下构建一个相对可信的计算环境的理论和方法。因此，可信环境体系结构的形态、如何建立信任链并且传递和管理信任关系、如何构造可信的安全多方计算环境、如何评价一个计算环境的可信程度成为必须面对的问题。主要研究内容包括：可信环境的数学理论与信任传递理论、可信计算环境构造机理及方法、可信计算环境测评。

2.3.4　可信软件演化

可信软件的演化最终目的就是强调软件系统在结构修改和功能调整期间仍能够持续提供服务，并使可信软件在维护、演化、自适应等相关概念基础上衍生的一种粒度更细、关注点更集中的概念，其目的是在不中断软件系统所提供服务的前提下，提升软件系统的可信性和完整性，从而提高软件系统适应需求的能力。通过相关文献查阅，选择以下三种有代表意义的可信软件演化方法来进行描述。

1. 基于服务组合的可信软件演化机制[33]

首先，为了保障基于服务组合软件的演化操作的可信性，提出了一个业务流程合理性保持的演化操作集。该集合中定义了流程结构的基本演化操作，并且证明了使用这些操作对确保演化后流程结构的合理性，从而避免了传统方法在演化后的复杂验证过程，如图 2-49 所示。其次，针对演化结果的可信性问题，特别针对可用性这一可信属性，给出了一种面向可用性保障的组合服务演化方法。该方法通过分析组合服务业务流程结构，使用基本演化操作集构造冗余路径来保证业务流程中“关键区域”的可用性。再次，针对组合服务动态演化实施过程的可信性问题，设计了一种组合服务演化中运行实例在线迁移方法，使其能够判断运行实例是否能够运行时迁移，并且可以准确地计算出迁移后的实例状态。最后，基于面向方面编程 AOP（aspect oriented programming）的思想，设计实现了一个支持动态演化的组合服务执行引擎以及相关的实验，如图 2-50 所示，并证明了可信的组合服务动态演化方法的合理性和有效性。

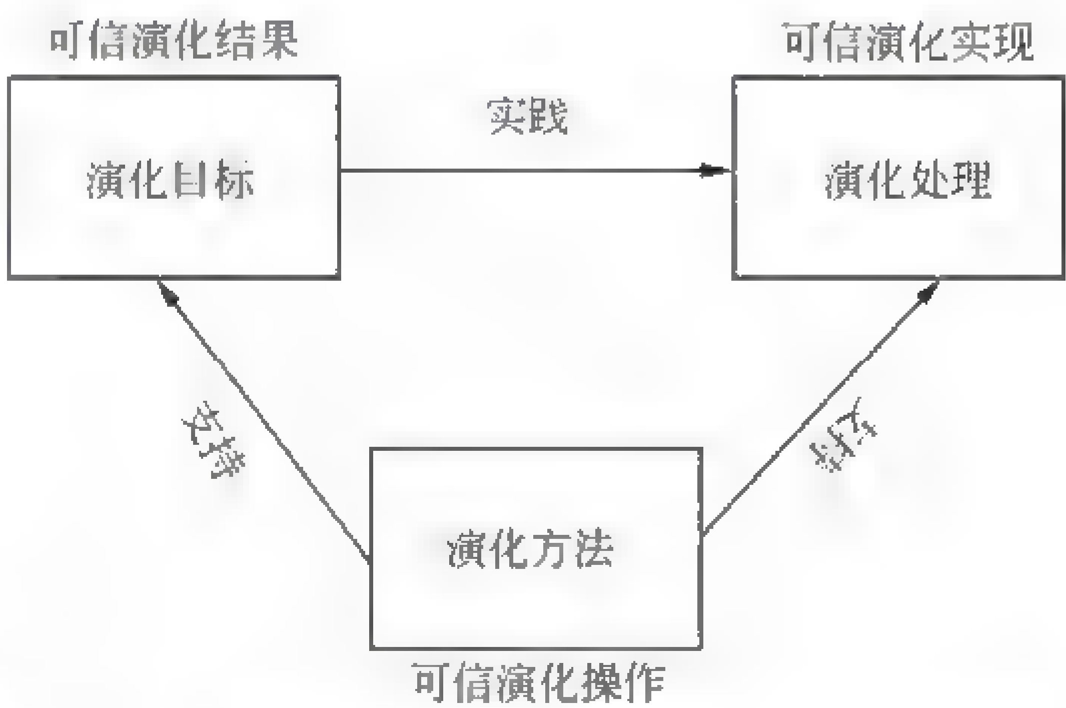

图 2-49　基于服务组合的可信软件动态演化研究思路

2. 一种支持软件可信演化的构件模型[34]

在该文中根据要支持软件适应能力的高效在线调整机制，即在软件工程层面有必要实现感知、决策和执行三者的关注点分离，并支持它们的在线重配置。针对这一挑战，在该文中提出了适应性构件模型 ACOE（Adaptive Component Model For Open Environment），图 2-51 所示是 ACOE 软件的构造和组装。ACOE 引入感知构件和行为构件类型，在构件组装层面引入抽象“环境变化-适应动作”关系的策略连接子，从而使得感知、决策和执行这三个关注点可以独立表达和封装。在此基础上，ACOE 基于动态软件体系结构（Dynamic Software Architecture，DSA）技术实现这些关注点的在线重

配置。上述机制使得第三方可以通过对各个关注点的有选择性更新来细粒度地调整软件适应能力，在环境超出开发阶段的预期时保证软件的可信。

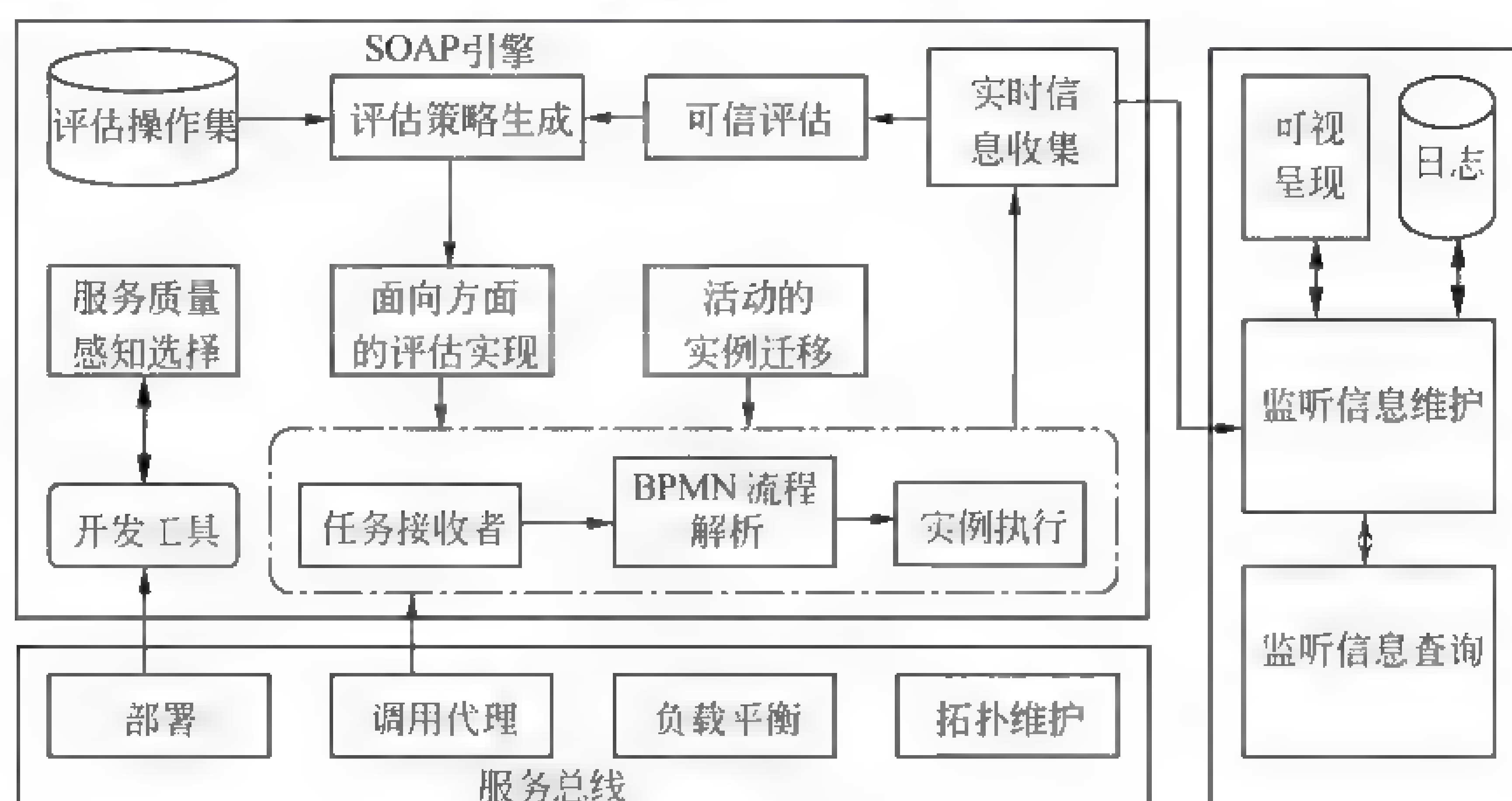

图2-50　SOAREngine的体系结构

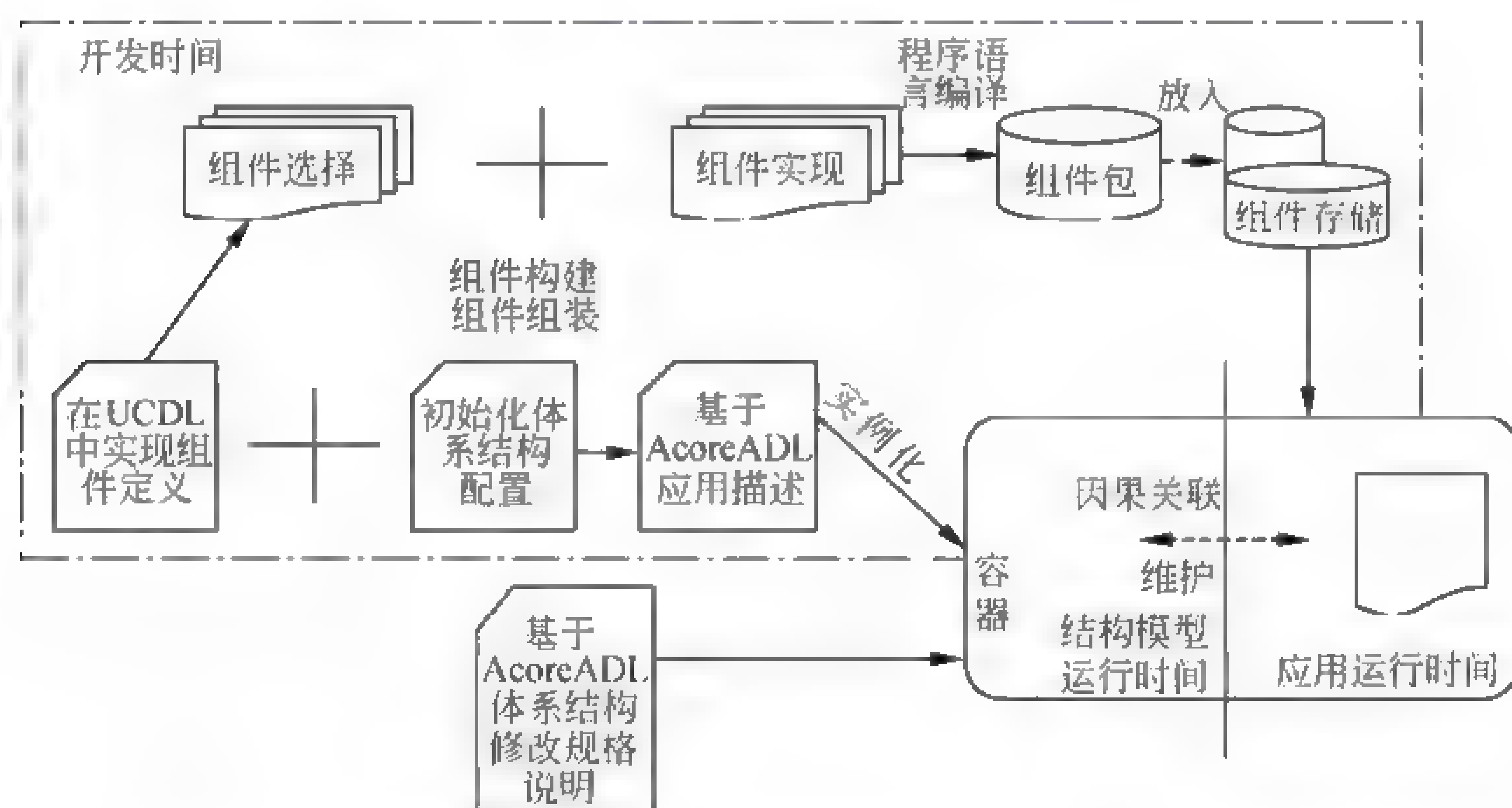

图2-51　ACOE 构件的构造和组装

3. 基于本体的可信软件演化框架模型[35]

在该文中，作者在服务和构件技术上开展了可信软件演化机制的研究，指出软件系统运行于一定的开放环境，其应用场景是：用户使用系统，系统服务用户，并在此过程中采取各种方式利用外部环境（主要包括第三方实体及其相关信息）来提高服务质量。而整个软件深化机制结构分为环境设施部分、协同系统部分和目标驱动部分三部分。

可信软件演化的核心是通过感知和评估环境变化对软件可信性的影响，从而来控制软件可信性的增长。为解决语义问题，该首先采用本体空间的方法。该本体空间既能描述问题空间子模型，也能描述解空间子模型；既能用于刻画运行系统的子模型，也能描述运行环境的子模型。并在已有的本体空间内，通过抽取本体库中的概念，建立环境要素到软件行为之间的映射，以及相应的推理过程，再通过规约制导和人工分析生成能够有效提升软件可信性的演化方案；最后在软件全生命周期内通过工具来实施具体方案。其演化结构如图2-52所示。

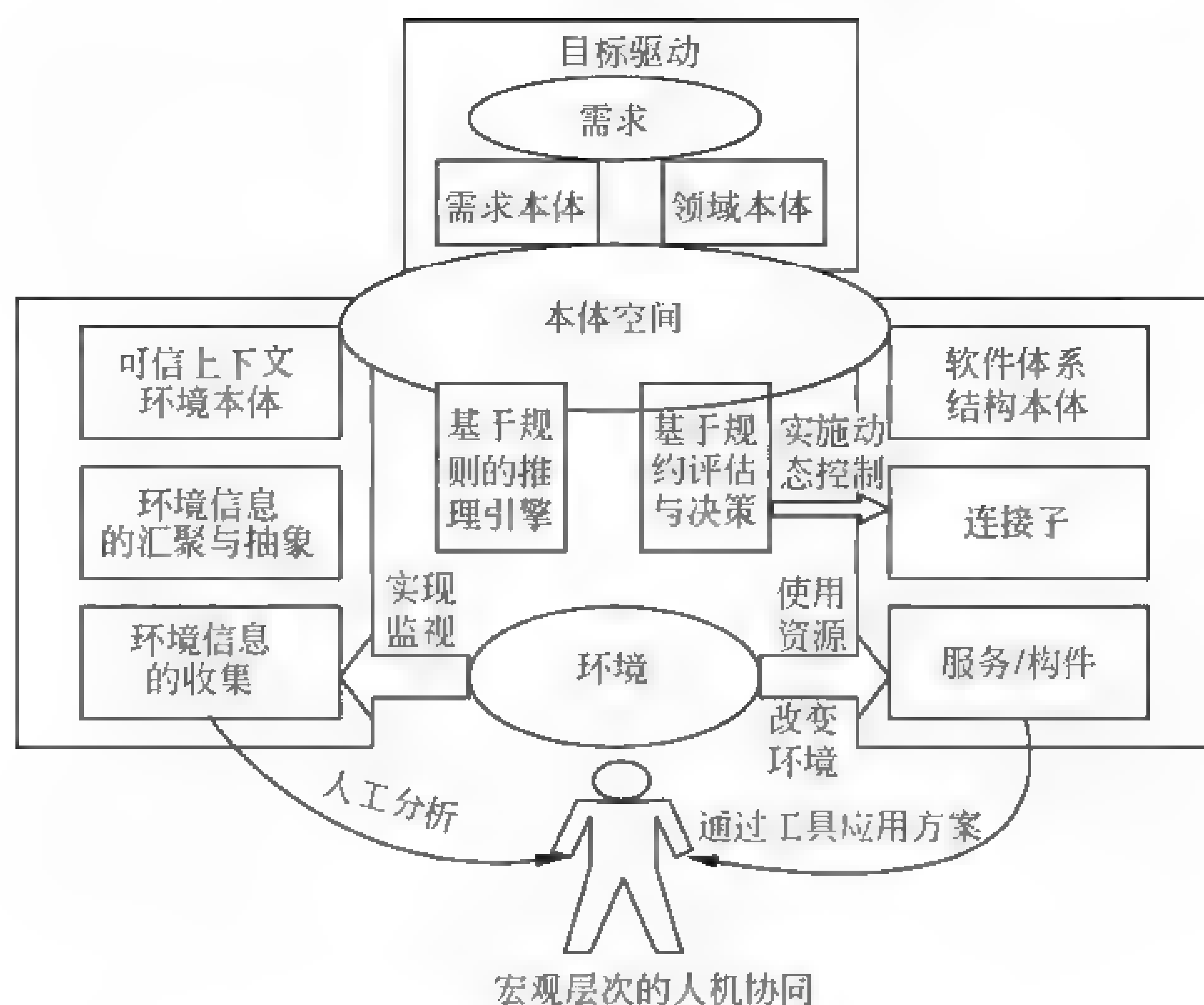

图 2-52　基于本体的可信软件动态演化框架

2.3.5　可信软件度量

目前，关于软件可信性的度量研究主要集中在软件的可靠性度量和安全性评估两个方面。并且 Y.K.Malaiy 等认为，软件可靠性模型是软件可靠性度量的主要手段[36]。M.Ohba 依据建模对象把软件可靠性模型分为两大类[37]：静态模型和动态模型。其中，静态模型利用软件复杂度来确定软件缺陷数，它的建模对象是软件的各种复杂性参数，包括：软件大小、复杂度、技术层次、决策数目等；它的特点是不需要进行软件测试即可进行软件缺陷估计。而动态模型是研究者提出最多的一类模型，以与运行时间有关的数据或信息为建模对象，同时也可以包括与运行时间无关的数据或信息。动态模型利用测试过程中获得的软件缺陷失效时间，或在一段时间内软件的失效数来估算整个软件的缺陷数和下一个软件失效发生的时间，或者其他一些与软件失效有关的数据。这类模型主要包括马尔科夫过程模型、非齐次泊松过程（NHPP）模型和贝叶斯模型等。

根据模型应用的阶段，又可分为两类：一类是面向整个开发过程的建模；一类是面向后期测试阶段的建模。第一类动态模型中最常见的是 Rayleigh 模型。Rayleigh 模型是一个形式化的参数模型，主要用来预测开发工作完成后软件中潜伏的缺陷。第二类动态模型一般又称作软件可靠性增长模型，通常以正式测试过程的数据为基础。该类模型主要是在开发后的测试阶段。同时，由于软件的可靠性随着故障的发生、缺陷的修复、测试时间增长而增强。这时随着软件可信性概念的引入，依据建模运用的数学方法，软件可信性模型又可以分为两大类，一类是基于经典概率统计学的参数估计模型，另一类是计算数学的非参数估计模型[38]。

下面举三种常见可信度量模型[31]。

1．TCG 在信任链中采用的是度量数据完整性的静态度量

在可信计算的信任链中理应度量可信性，但是，由于可信性目前尚不容易直接度量，这

时 TCG 在信任链中采用的是度量数据完整性，而且是通过检验数据 Hash 值的方法来度量数据的完整性，且完整只是可信性的一个方面。因此，TCG 的信任链测量是不完善的。

2．一种基于软件行为的动态完整性度量方法

基于软件行为的动态完整性度量模型，是通过分析可执行文件或源代码的 API 函数调用关系得到软件的预期行为，并建立软件预期行为描述集，并发布后对软件进程的实际 API 函数调用行为进行监控，如果在软件执行过程中，软件行为符合软件预期行为描述集中的相关规则、软件行为认证码，则认为软件是可信的，否则说明软件不可信，此时应该对该软件进行控制。

3．基于信任链的系统数据完整性多次度量

信任链是可信计算的关键技术，通过信任链的作用来确保系统数据的完整性，从而提高系统的可信性。但是，早期的信任链只是在平台启动时才进行一次完整性校验。这对于频繁开机和关机的 PC 来说是基本可以的。但是，对于服务器这种一旦开机后就长时间不关机的计算平台来说是远远不够的。用户很难相信开机时几分钟的数据完整性校验，但能够确保服务器几年的数据完整性。因此，需要能够反复进行系统完整性校验的信任链机制来实现系统数据完整性多次度量。

2.3.6 可信软件技术

可信软件技术就在可信软件原理、方法支撑下，实现按需求的可信软件系统，主要包括可信软件体系结构，实现可信软件的技术，以及支持可信软件二次开发的支撑平台。目前这些技术主要包括互联网的服务计算相关方法和技术[33]、构件技术、面向方面的方法等。

ISO/IEC 15408 标准关于可信性的描述：一个可信的组件、操作或过程的行为，在任意操作条件下是可以预测的，并能很好地抵抗应用软件、病毒以及一定的物理干扰造成的破坏。

可信计算机组织认为，一个实体总是按照其设定的目标所期望的方式运行，则这个实体是可信的。

Algirdas 等认为参考文献[39]，传统软件的可信性主要包括安全性和可靠性两个方面。王怀民等认为参考文献[40]，如果软件系统的行为总是与人们期待的愿望一致，那么该软件系统是可信的，并且进一步将可信性概括为身份可信和能力可信，其中，身份可信的核心是基于身份确认的访问授权与控制，能力可信的核心是软件系统的可靠性和可用性。

从这些不统一的可信软件的定义可以看出，要实现可信软件的软件系统，技术是多样的、复杂的。而软件可信性又可以分为基础可信和扩展可信两个部分，其中基础可信是指运行过程可信和结果可信，扩展可信是指可信性面临的风险程度及可控程度，一个可信的软件应该同时满足基础可信和扩展可信。运行过程的可信是通过可靠性和可用性来反映，运行结果的可信通过完整性来反映，可信性面临的风险程度通过保密性以及交互性来反映，风险可控程度通过防危性和可维性来衡量。其中，运行过程可信是基础，没有可信的运行过程，可信性将成无本之源；结果可信是核心，失去可信的结果，可信性度量将是一个伪命题；可信性面临的风险及其可控程度则是可信性保障能力的反映，对软件可信性的判定具有重要的参考作用，图 2-53 是软件可信性结构模型图[38]。

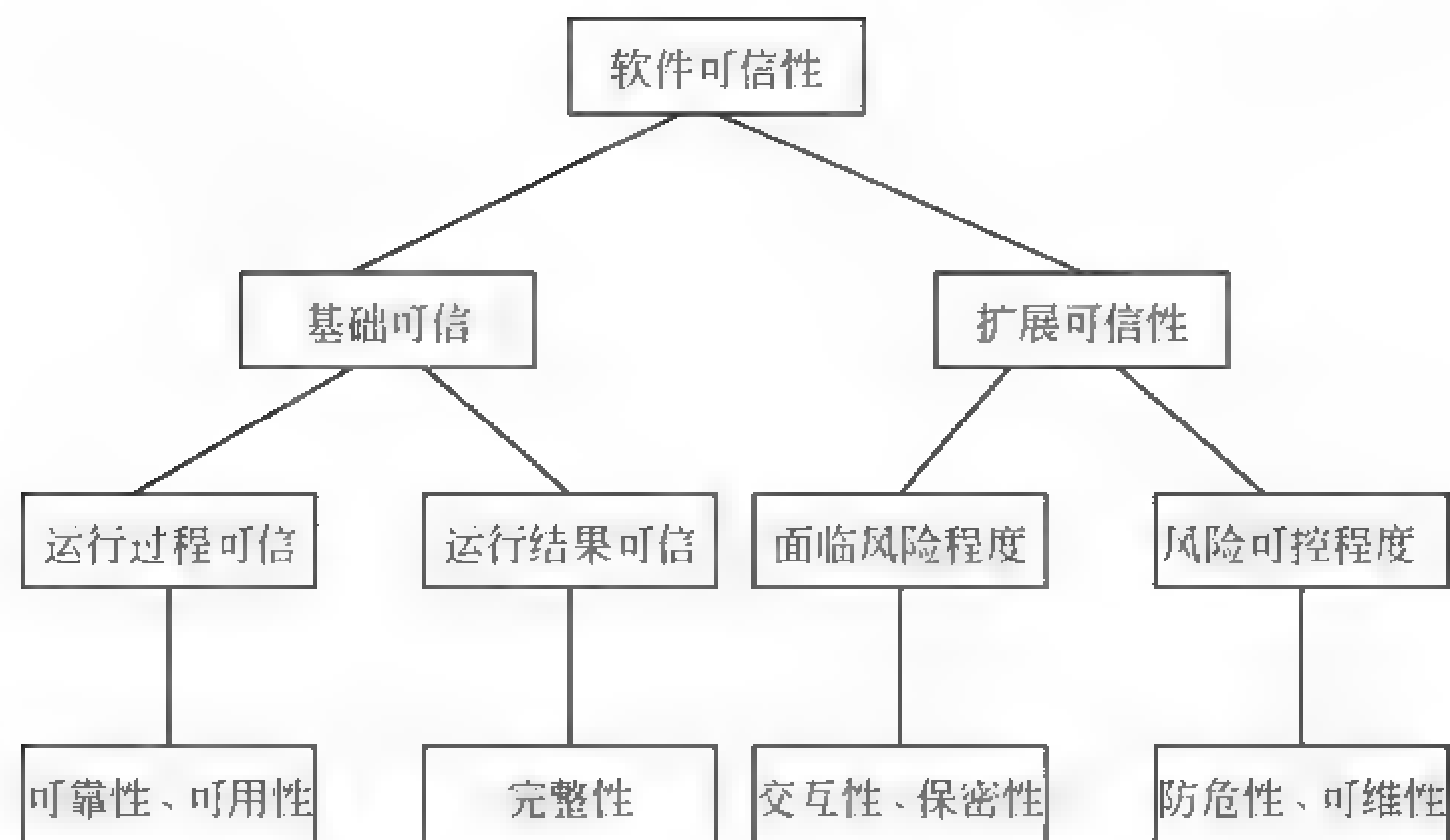

图 2-53 软件可信性结构模型图

并且从图 2-53 可知，从软件可信性问题研究的演变轨迹可以看到，软件的可信性问题是从最初的软件可靠性问题，随着软件应用环境的逐渐开放化、网络化、复杂化的结果。软件可信性度量研究是计算机技术应用中产生的软件信任问题越来越全面、越来越复杂的要求和体现[38]。这也可以说明软件的可信性是需要考虑需求与软件可靠性、健壮性、可用性、安全性等相关的因素的影响。下面就根据此概述一种可信软件体系结构和可信软件检测方法。

1. 支持运行监控的可信软件体系结构设计方法[41]

通过引入面向方面的软件体系结构设计方法及其相关概念，提出一种支持运行监控的可信软件体系结构设计方法。并在支持运行监控的可信软件构造模型 TSCM（Trusted Software Constitution Model Based on Monitoring）的基础上，利用一种面向方面的体系结构描述语言 AC2-ADL 描述具有监控能力的软件体系结构，试图为分析和设计具有监控能力的系统的软件体系结构提供一种有效的解决方案，图 2-54 所示是基于监控的可信软件构造模型 TSCM。其中 TSCM 是一种提高软件可信性的软件生产技术。该模型认为对软件实施有效的监控以获得软件的运行状态，并通过将监控结果与预期行为进行比较，可以有效地掌握软件行为的可信程度，且有助于准确分析和定位软件故障。AC2-ADL 除了传统的 ADL 中所包含的概念元素以外，还引入了方面构件（aspectual component ） 和方面连接件（aspectual connector） 作为体系结构层的第一类元素。并且 AC2-ADL 结合一种时序逻辑语言 XYZ/ E 对各种构件的行为进行逐步求精，并结合面向方面的软件开发中特有的性质和概念，扩展了 XYZ/ E 中类其中主要元素的相关语法。

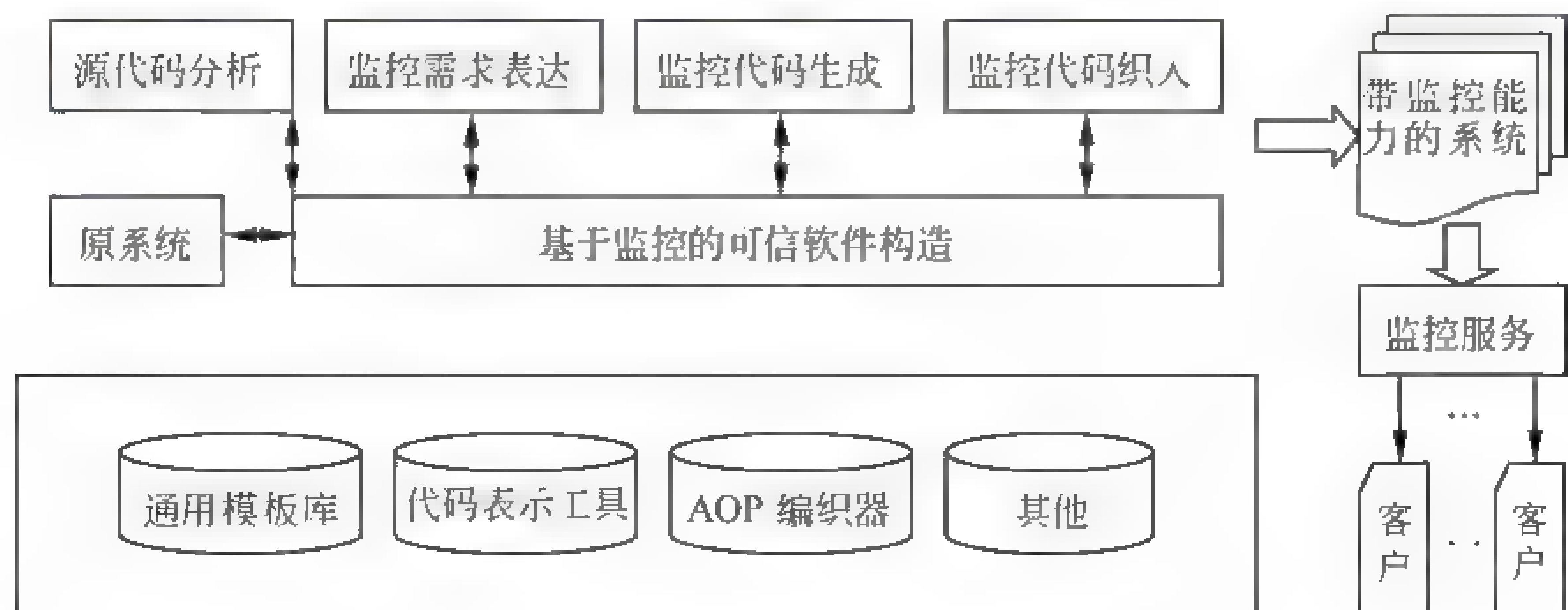

图 2-54 基于监控的可信软件构造模型 TSCM

最后并通过面向方面的技术（AOP）来实现可信软件成为一种重要选择策略之一，该方法充分体现了关注点分离（Separation of Concerns，SoC）的思想，通过使用方面（aspect）及其相关概念封装和管理分布在软件开发周期各个阶段的横切属性，有利于提高软件产品的质量，并减少适配、演化和维护的开销。因此，在软件体系结构层，使用侧面建模和管理软件的可信性，可进一步提高软件体系结构设计方案的可理解性、可演化性和重用性。

2．基于 QFD 和 TRIZ 的可信软件技术冲突解决方法[42]

为解决软件开发中的技术冲突及提高软件的可信性，提出了基于质量功能展开（QFD）和发明性问题解决理论（TRIZ）的可信软件技术冲突解决方法，其中 QFD（Quality Function Deployment）是连接用户与技术人员的有效方法，并以用户需求为依据，通过质量屋（House of Quality）将用户需求转化为产品开发过程的一系列技术特征，在开发初期就以产品的质量和适用性实施全方位保证的系统化方法；同样，QFD 也可将软件的可信性需求逐步映射至软件开发的整个过程，保证可信性需求在软件的开发过程中得到正确而一致地实现，进而提高所开发软件的可信性。但 QFD 在解决可信软件设计中存在技术冲突的相关关系：负相关，正相关和不相关，因此，这些可以用发明性问题解决理论（Theory of Inventive Problem Solving，TRIZ）中技术冲突定义；并进一步说明呈负相关关系的技术特性一，其本质就是产品开发中的技术冲突。因此，使 QFD 与 TRIZ 结合起来，借用 TRIZ 的创新原理来解决可信软件设计中的技术问题。

因此，针对可信软件中的技术冲突问题，在参考文献[42]中的作者首先构建可信软件规划质量屋，在软件开发过程中引入用户视角；其次，重点分析质量屋的技术特性自相关矩阵中呈负相关的技术特性，尝试用 TRIZ 的发明创新原理予以解决，获得多个可行的创新性解决方案；在综合专家意见方面，考虑了专家语言表达的模糊性、不确定性和多粒度多语义的情况，用近年来最新发展的基于语言信息的决策理论来评估和选择最可行的解决方案。图 2-55 就是基于 QFD 和 TRIZ 的可信软件技术冲突消解模型。

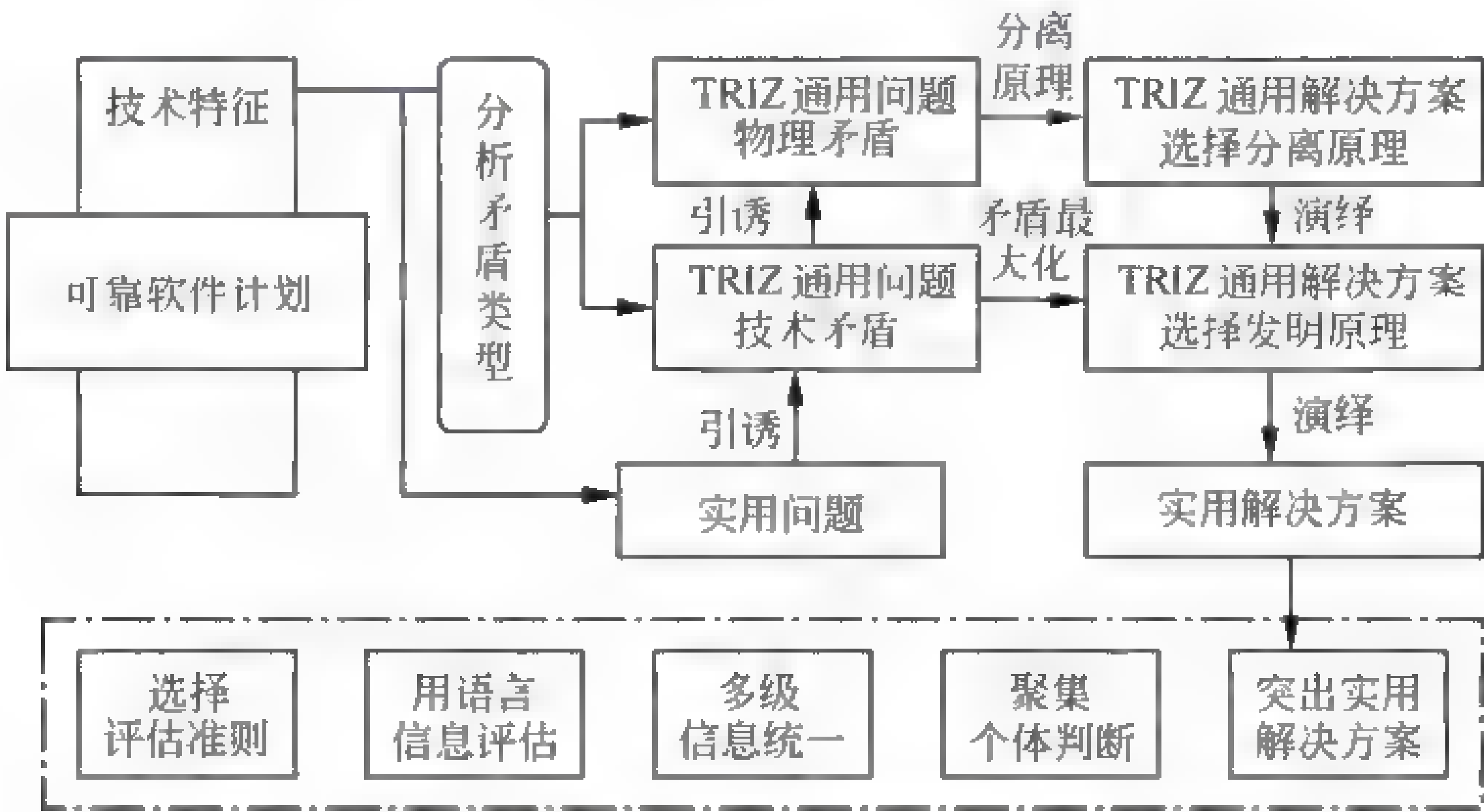

图 2-55　基于 QFD 和 TRIZ 的可信软件技术冲突消解模型

2.3.7　可信研究进展

国内外的国家的政府组织、跨国公司、大型科研机构近年来逐步认识到可信软件研究的

巨大价值和应用景，并可以给国家和各团体的经济和安全带巨大的效率和保障，纷纷有针对性地提出了相关研究计划。美国《国家软件发展战略（2006—2015）》将开发高可信软件放在首位，提出了下一代软件工程的构想；美国国土安全部 2006 年启动了 Software Assurance Program，目的是改进可信软件产品的开发与部署；并且美国自然科学金也投入了大量资金研究可信计算，并与美国大学和 IT 公司建立建立合作研究计划。国际上著名的研究项目还有美国自然基金会新近启动的 Science of Design 计划，NASA 在卡内基梅隆大学支持的 hdcc 项目，DARPA 资助的 OASIS，德国教育研究部资助的 Verisoft 和德国研究联合会（DFG）资助的 AVACS 项目，英国的 INDEED 和 DIRC 研究项目等。但无论是国外还是国内，在构建可信计算环境方面都呈现出技术超前于理论、理论滞后于技术的状况。到目前尚没有公认的可信计算理论模型供人们全方位使用。

1．在国外研究情况

1983 年美国国防部制定了世界上第一个《可信计算机系统评价准则》TCSEC（Trusted Computer System Evalution Cirteria）。1993 年，IBM、HP、Intel、微软等著名 IT 企业发起成立了可信计算平台联盟 TCPA（Trusted Computing Platform Alliance）。2003 年 TCPA 改组为可信计算组织 TCG（Trusted Computing Group），标志着可信计算技术和应用领域的进一步扩大。TCPA 和 TCG 已经制定了关于可信计算平台、可信存储和可信网络连接等一系列技术规范。欧洲于 2006 年年 1 月启动了名为“开放式可信计算（Open Trusted Computing）”的可信计算研究计划，已有 23 个科研机构和工业组织参与。微软提出了代号为 Palladium 的可信计算计划，推出的新一代操作系统 VISTA 支持可信计算机制。

同时，Microsoft 公司根据可信计算发展趋势，于 2002 年 5 月提出高可信计算概念，从目标、手段、实施三个方面对高可信计算进行了解释。从最终用户实际需要出发，其目标包括三个方面：安全性、可靠性、完整性，而实现这些目标的手段必须从商务和工程方面进行考虑，遵循的策略包括：安全策略、可用性策略、隐私保护策略、可管理性策略、准确性策略、易用性策略。

2．在国内研究情况

2000 年 6 月月武汉瑞达公司和武汉大学合作，开始研制安全计算机，2003 年年研制出我国第一款可信计算平台模块 TPM （J2810 芯片） 和可信计算平台，并通过国家密码管理局认证。2004 年 6 月由瑞达公司和武汉大学联合在武汉大学召开中国首届可信计算平台（TCP）论坛；同年 10 月，在武汉大学召开了“第一届中国可信计算与信息安全学术会议”。2005 年联想公司的 TPM 芯片（恒智芯片） 和可信计算机相继研制成功；同年兆日公司的 TMP 芯片也研制成功，这些产品也都通过了国家密码管理局的认证。2006 年，我国进入制定可信计算规范和标准的阶段，在国家密码管理局的主持下制定了《可信计算平台密码技术方策》和《可信计算密码支撑平台功能与接口规范》两个规范。2007 年，在国家信息安全标准委员会的主持下，我国开始制定一系列的可信计算标准，包括芯片、主板、软件、可信网络连接等标准，国家自然科学基金委也启动了“可信软件重大研究计划”。深圳中兴集成电路公司的“可信计算机密码模块安全芯片”和联想公司的“可信计算密码支撑平台”通过国家密码管理局的认证。2008 年，中国可信计算联盟（CTCU）成立，北京兆日公司的“可信计算机密码模块安全芯片”和“可信计算密码支撑平台”、深圳中兴集成电路公司的“可信计算密码支撑平台”通过国家密码管理局的认证；在国家“863”计划项目的支持下，武汉大学研制出我国第

一款“可信PDA”和第一个“可信计算平台测评系统”。2009年，瑞达公司的“可信计算机密码模块安全芯片”通过国家密码管理局的认证，基于这一新芯片的可信计算机也推出上市[31]。

而对于可信软件来讲，则离不开可信环境来支撑，随着软件网络化日趋明显，这种可支撑的环境越来越重要，这也是可信软件“生存”的基础。可信软件的基本要求是具有正确性、可靠性、安全性、生存性等特性，但这些特性只能表达可信性的一些基本的属性，不能简单说具备了这方面的特性就说该系统就具有可信性。并且这些特性来源的基础就是能得到可信计算环境的支持。这是因为[30]：

（1）软件系统越来越复杂，软件可信意味着软件行为可信、环境可信和使用可信等不同层次的可信要求，而局部的可信并不一定导致全局的可信。系统的可信性属于涌现类的性质，如何从整体上度量、获得并保证可信性将是非常困难的。

（2）不同可信属性之间可能彼此有冲突，并且不同层次之间也可能会有冲突，如何最优化地协调与取舍也是一个关键问题。

（3）当软件可信性成为研究目标之后，必然要针对“可信”性质建立分析、构造、度量、评价体系，使得可信性能够在软件生产活动中被有效地跟踪控制和验证实现，这也对现有的计算理论与技术体系提出了挑战。需要强调的是，要达到软件可信的目标，需要对软件系统开发的整个生命周期，需求分析、可信算法设计、软件设计与实现、测试与验证、运行维护等阶段进行全面而统一的研究。用户对软件可信性的认可还有一个积累和沉淀的过程；而在软件运行过程中，软件可信的可演化特征也是现有静态分析、测试技术无法应对的。

针对网络软件的可信演化机理，在参考文献[43]通过分析互联网软件的可信内涵及互联网应用的基本特征出发，研究了网络化软件的可信机理。指出在开放、动态的互联网环境下，软件的身份可信、能力可信都面临新的问题和挑战；以及基于自主协同运行模式的新型网络应用需要对软件群体行为进行有效约束，以保障其行为可信。这不仅需要从传统的安全、可靠、可用的角度提高软件个体的可信性，还需要从软件群体行为可信的角度对软件的协同行为进行约束。因此，在此基础上，给出了一种面向互联网虚拟计算环境的互联网软件可信概念模型，并提出集身份可信、能力可信和行为可信为一体的网络软件可信保证体系，并以跨域的授权管理机制、高可用的服务保证机制和面向自主协同的行为激励机制为突破口，研究和探索互联网软件可信问题的技术途径。

以上概述了现阶段可信软件的研究基础、实现方法及面临的问题，这为基于可信机制的开源软件的构架方法提供可信机制。

2.4 协同软件构架方法

协同软件（Collaboration Software）是打破资源（人、财、物、信息、流程）之间的壁垒，实现资源有序掌握、协调和优化运作，达成资源的最优化利用，从而提高软件协同能力。CCW Research将协同软件定义为：协同软件是个人或组织用来实现沟通和协作的应用软件，是一种管理软件，必须体现协同化的管理；而协同化管理就是将企业各种资源（包括人、客户、财物、信息、流程）关联起来，为能够完成共同的任务或目标而进行的协调或运作；同时，协同软件自身蕴含特定的管理思维或管理模式，是与其他管理软件的主要区分特征。简单将

协同软件概述为：是指那些以团队协作为目标的协作软件工具，主要包括群组协作管理，如：工作流管理、项目管理、各种通信软件等。而协同计算（Collaboration Computing）则源于计算机支持的协同工作（Computer Supported Cooperative Work，CSCW），是指在地域上处于分散状态的一个群体借助计算机和网络技术，相互协作共同完成一项任务。研究工作主要包括协同工作系统的建设、群体工作方式、支持群体工作的相关技术、应用系统开发等。使得协同计算是多学科交叉和支持的研究领域，它将不同学科中共同存在的协同现象抽象出来，作为研究对象，以便更有效地促进社会群体间有目的的交互和协作。但协同的实现仍还是以软件产品和过程活动为中心来实现软件协同的模式。

采用协同构架模式来指导面向开源软件的软件模式主要是将协同的基本原理、方法和技术应用于面向开源软件的构架中，指导正确使用开源软件来研发不同的软件系统。从而使面向开源软件的软件开发形成一种新的软件工程模式，也对提高软件复用率、实现软件群体智慧创造工程性指导。

2.4.1　协同软件概述

据 Gartner 统计分析，从 2003 年开始，全球范围的协同软件已成为用户应用软件采购最大热点，位居信息化应用软件首选，到 2005 年全球协同软件市场的营业额将达近 500 亿美元，到 2006 年协同软件市场规模将赶超 ERP。在国内，通过 CCW Research 报告指出：2005 年，中国协同软件市场达到了 38.2 亿元，其中协同工具软件市场为 0.8 亿元，协同平台软件市场为 7.7 亿元，协同应用软件市场达到了 29.7 亿元。预计 2010 年协同软件市场达 56.4 亿元，并且认为，2010—2013 年，协同软件在中国市场将以平均 33.7%的年复合增长率加速发展，到 2013 年市场总额将达到 132.2 亿元。在 2009 年协同软件市场加速发展，销售总额达到 41.4 亿元，与 2008 年相比增长了 40.1%。其中协同办公软件市场销售总额达 28.1 亿元，与 2008 年相比增长了 48.8%；协同平台软件和协同工具软件的市场销售额分别为 11.4 亿元和 1.9 亿元，与 2008 年相比分别增长了 24.9%和 23.2%。随着制造业、政府行业在协同软件市场中的地位不断增强，市场份额还会不断提升。同时 CCW Research 研究认为：协同软件的出现、发展可以追溯到上世纪 90 年代，并已经了以世纪 90 年代中期、后期及三个基本阶段，如表 2-18 所示。

表 2-18　协同软件发展历史

阶段	起步阶段	功能扩展发展阶段	广泛应用阶段
时期	20 世纪 90 年代中期	20 世纪 90 年代后期	21 世纪初期
功能特点	群件/邮件	知识管理、项目管理、联系人管理等	业务流程、应用和网站脉络，用户驱动的内部协作环境
厂商	Lotus、Novell、Microsoft、IBM、Oracle	IBM Lotus、Microsoft、Open Text、Groove 等	IBM Workplace、Microsoft Office System、Oracel Collaboration Suit 10G 等

2.4.1.1　协同软件分类与作用

信息化软件也可以分为两大类：ERP（Enterprise Resource Planning）软件系与协同软件

系。其中，ERP软件系包括：财务软件、物流软件、CRM（Customer Relationship Management）、HR（Human Resources）、库存软件甚至各种行业性业务管理软件等，其作用主要是帮助企业业务管理；协同软件系包括：协同OA（Office Automatic）、HR、CRM、绩效、网络、门户、IM（Instant Messaging）、邮件等。

1. 协同软件的分类

现根据功能划分角度将协同软件分为三类：

（1）协同工具软件：是指独立的、功能相对简单、用于相互间沟通的软件，包括独立的企业电子邮件系统软件、企业IM软件、企业VOIP系统软件、企业IP电话会议系统软件、企业统一消息系统等。

（2）协同平台软件：是指提供一个协同化的架构（包括开发工具和开发环境），可以让用户或独立软件开发商在此基础上开发最终应用的平台性软件。这些产品主要包括IBM Lotus Doimio、微软的Sharepoint Server等软件产品。

（3）协同办公软件：是指帮助企业实现协同化管理和办公的最终应用软件，主要包括综合通信、协作区（含团队协作等）、联系人管理、工作流、文档管理等类型。而部分协同办公软件在是在第三方协同平台软件基础上开发的，另外也有部分在系统软件或中间件上直接开发的。当然，部分企业管理软件在其开发和应用的过程中，逐步接受或采纳了协同的理论和技术，则该部分软件也属于协同办公软件范畴，包括协同OA/协同政务、协同知识管理、协同CRM、协同销售、协同HR等类型。

2. 协同软件的作用

CCW Research进一步研究认为协同主要包括四个资源元素：人、信息、流程以及应用。并且人是协同的核心，体现以人为本的管理思想；信息协同实现信息资源的高效整合与共享；流程协同实现相关流程的无缝衔接；应用协同实现不同应用系统之间的优化整合。信息、流程、应用的协同是为实现人的高效协同而服务的。同时指出，协同软件具备以网络为基础、以流程协作为主、以人为本等三大特征，在企业运行中的应用主要体现在对信息的高度共享、对业务应用整合、对资源的调配和优化等三个方面。其主要表现在：

（1）借助沟通平台促成信息高度共享：综合沟通平台通过提供综合通信、文档审批和交换、视频会议、即时通信等群组协作环境，最大限度地为人员之间沟通提供良好的IT环境。在提供群组协作的同时，协同软件还通过综合文档管理、动态数据处理与企业信息门户，高效整合分散于企业内外的各类文档、数据资料与其他信息，挖掘信息的内在关联，增加信息可理解性与再加工能力，促进信息共享协同。

（2）作为平台整合业务应用：随着企业信息化建设的深入，在企业中存在ERP、CRM、销售管理、财务管理、人事管理等多种应用模式。协同软件是企业将这些应用进行整合和统一展现的平台。是对应用软件的整合，主要通过展现层面，通过在各个应用系统与协同软件之间建立相互联系，为企业的管理者、员工提供统一的处理各种事务的渠道和门户。

（3）建立团队协作环境，优化资源调配：在企业中，除个人之间的沟通之外，协同软件的另外一个广泛应用是提供团队协作环境。在企业中，大量的密切沟通来自项目组、各个部门团队，并且在团队成员之间在不断变化。协同软件除为团队成员之间提供良好的综合通信、信息协同等服务功能外，还可以团队之间的流程控制、审批、团队管理等提供良好的支持。企业内部各个部门，各个动态虚拟的项目团队，都可以通过协同软件组建与管理，并促进团

队工作效率的提升。

2.4.1.2　协同软件中国市场结构分析

由于协同软件逐渐实现了人、信息、资源三者间的沟通和流通，必能为市场带来很好的经济效益，根据 CCW Research 的研究报告从以下几个角度来分析协同软件在中国市场的情况。

1. 产品结构分析

2011 年、2012 年协同软件市场产品结构与 2008 年大体一致。协同办公软件仍然是市场的主力，且与 2008 年相比所占市场份额有所提升，达到了 67.8%；协同平台软件和协同工具软件所占市场份额则较 2008 年有所下降，分别为 27.5%和 4.7%。

2. 区域结构分析

从区域结构上看，华北、华东和华南区域市场排名前三，所占市场份额分别为 29.0%、25.4%和 22.7%，与之相对应，主要的协同软件厂商总部都分布在北京、上海和广东。与 2008 年相比，华北区域所占市场份额有所回落，华南、华东地区所占市场份额微幅上浮。

3. 行业结构分析

2009 年，协同软件制造业市场发展迅猛，占总体市场的比例达了 32.1%，与 2008 年相比增加了 12 百分点，是协同软件规模最大的产业市场。政府行业占市场总体的 16.8%，较 2008 年有大幅增长。流通业、金融业、电信业以 12.4%、11.7%和 7.5%的市场占比紧随其后增长。

2.4.1.3　协同软件中国市场特点分析

根据 CCW Research 研究报告指出，协同软件市场特点也逐渐呈现多样化、复杂化、成熟化。2009 年，协同软件产品核心功能应用得到了不断扩展完善，SOA 技术逐步落地（2005 年年初，我国首套 SOA（Service-Oriented Architecture）协同软件在上海问世，标志着我国协同软件赶上甚至部分超过国外水平。协同软件指那些以团队协作为目标的沟通协作软件工具，主要包括办公自动化、电子政务、工作流管理、知识管理、信息门户等应用，集成整合应用也发展迅速。主要表现如下：

1. 产品特点分析

核心功能应用得到了不断扩展完善。在满足企业对信息共享、跨部门文件流转、日程控制和管理等三方面的基本需求后，协同软件加强了信息门户、工程流程管理、知识管理、员工协作，目标和任务管理等功能，进一步满足了管理复杂的大型集团企业客户的需求。同时，对部分细分功能也进行了强化，如关系管理功能在支持内部通信簿管理的基础上，加强了对客户通信簿和客户文档资料管理的支持力度。

SOA 技术逐步落地，集成整合应用发展迅速。由于制造业、政府等行业在协同软件市场中的地位不断增强，市场份额不断提升，对业务灵活性和数据接口的要求将会更加苛刻。协同软件只有采用更加灵活的 SOA 构架，才能满足不断需求变化的要求。同进，伴随着 SOA 技术的逐步落地，可以将企业信息化看做以协同软件为切入口、以用户应用为主导的信息化平台建设，将其他各业务软件整合到平台上来。2009 年协同软件集成整合应用发展迅速，与 ERP、CRM、KM（Knowledge Management）等各领域应用软件形成了整合解决方案。

2. 渠道特点分析

目前国内协同软件的主要采购渠道仍然是软件服务集成商和软件代理商，两渠道的采购量占到了总采购量的六成以上，对销售增长的贡献度也无高于直销。但随着市场竞争的不断加剧，渠道争夺也愈演愈烈。建立完备的渠道体系，加强渠道代理商的扶持与培训已成为协同软件厂商争夺市场的必要手段。一般可以通过增强渠道的技术、服务能力，加强渠道管理，提高渠道忠诚度，拓宽渠道覆盖面来提供渠道建设。

3. 服务特点分析

协同软件产品的自身属性决定了对服务有着较高的要求，重产品轻服务的短视行为注定难以在激烈的市场竞争中长期生存。协同软件自身蕴含特定的管理思维或管理模式，是企业解决内部管理问题、提升企业管理水平的重要手段。而在协同软件应用方面包括评估咨询、实施、维护及培训等阶段；协同软件厂商需要具备为客户提供管理咨询、实施意见的能力，同时需要保证后续服务的优质完善。

4. 用户需求分析

国内企业在采购协同软件时主要关注产品是否遵循标准，产品的体系结构、产品技术先进性、平台无关性、产品或解决方案的成熟度。同时，对于已经部署过业务系统的用户，若新协同软件不能带显著的业务提升，将会影响用户采购的愿望；而对于从没有部署过业务系统的用户，协同软件的吸引力主要体现在技术先进性、平台无关性等方面，当然还会考虑到价格、是否需要二次开发、实施时间以及后续服务的问题。

2.4.1.4 协同软件发展趋势

从性能角度来看，协同软件在高端市场将向高应用、高易用、高兼容和高扩展性方面发展，而在低端市场上，协同软件将多功能、低成本、易维护和强易用等方向发展。并且根据客户的需求和技术展，协同软件将呈现门户性、整合性、移动性和服务性等四个方向。

1. 门户性

是指协同软件为企业提供一个统一管理的门户平台，用户可以将其现有或即将部署的其他业务系统的数据和信息通过此平台进行共享，不同部门之间通过该平台来进行沟通协作。

2. 融合性

协同软件的范畴有逐渐延伸，从基本的OA软件和工作流软件扩展到以沟通为核心的多种功能的集合，包括即时消息、Web会议、VOIP，团队网上社区、呼叫中心等。未来几年，协同软件将融合更的功用，如电子商务等。

3. 移动性

随着移动基础设施的完善，移动办公将会加速发展。支持移动设备的协同平台软件，或协同工具软件将是各厂商的必争之地。

4. 服务性

SaaS和云计算的热度持续攀升，在安全可靠性得到保障的情况下，协同软件势必会延展到云端，这样企业不必再耗费精力进行软件的部署和维护，这些将由协同软件供应商和运营商去操作和实现部署。

2.4.1.5 协同软件的优势与不足

采用协同软件思想构架企业级的软件系统具有管理、需求和功能相结合的优势，这样不

但可以做到管理与计算机技术结合形成具备协同多方工作的软件，也可以实现管理信息化。这些优势主要表现以下三个方面：

1. 软件功能的优势

协同软件通过流程规范、信息共享、人员协助建立了虚拟化企业组织，通过各种事件的执行，将企业制度和文化渗透在这个虚拟的组织中，能够为企业员工提供轻松、高效的办公环境，更好地解决运营过程中繁琐的，使得管理层能够有更多地时间考虑企业的发展、战略规划。

2. 软件技术的优势

在技术上，协同软件的应用模式从 P2P 转向 B/S、web、ERP、Java 等，并且采用 CSCW、SOA、EAI 和 EIP 等主流技术是目前主流的软件应用模式，不论是在人机交互或者是界面设计上都占有优势，并且操作简单，流程性好。

3. 应用领域的优势

在应用上，协同软件在 CRM、ERP、SCM、KM&CC、BPM 等多个领域都能发挥作用。这就说明协同软件所适用的范围不仅仅是面向某类企业，而是面向所有的协同办公，包括电子政务等。表 2-19 所示是协同软件与其他软件的比较。

表 2-19　协同软件与其他软件的比较

	技术特点	软件组成	应用范围	核心内容	优点	缺点
协同软件	应用模式从 P2P 转向 B/S、Web、ERP、Java 等	内外部信息门户、协同交流平台、办公管理平台和移动办公平台	行政办公、电子商务、电子政务	以团队协作为目标的协作软件工具	将管理、用户需求和功能有机整合	不易随管理变化，而完成功能变化，也不易完成以需求的定制。如何利用这些技术和业务组件，构建出高适应性的协同支撑平台，在此基础上扩展出各种企业的协同应用
OA	Intranet、Internet 技术 B/S 系统架构 JAVA、J2EE、WebService 技术	数据层之间实现数据沟通	日常行政管理、各种事项的审批、办公资源的管理、多人多部门的协同办公、以及各种信息的沟通与传递	提高内部信息交流、共享、流转处理	提高日常办公效率，有效实现无纸化办公，快速实现信息传递协作办公	当功能增多时，面向用户层界面复杂，不易操作
CRM	Java 为主流，其次是 VB、JSP 等，采用 SQLSERVER、DB2、B/S 结构	应用业务集成，业务数据分析和决策执行	销售管理模块、市场管理模块、客户服务模块	关注重点由提高内部效率向尊重外部客户转移，核心在于关怀客户	提高客户之间的关系，将有效把一种便于把握和理解的关系实现信息化，从而提高客户的忠诚度	难有效把握销售、市场和客户间的流程控制
ERP	C/S 体系、数据库、面向对象技术、图形用户界面、第四代语言 4GL、网络通信	流程作业管理；产品数据管理；管制报告；决策支持	财务管理模块、生产控制管理模块、物流管理模块、人力资源管理模块	各业务系统之间数据高度共享，业务流程自动化	将企业有关的信息完成进行整合管理，提高企业信息化程度	不易扩展，用户需求变化而无法有效的实现更新

2.4.2　协同软件原理

协同软件的研究可以包括计算机科学与技术、认识科学、心理学、社会学、行为科学等诸多领域。其中，在计算机科学与技术中，相关协同软件的领域主要包括分布式系统、网络和通信技术、人际交互技术、多媒体技术、人工智能等。在体系结构、协同透明、会话管理、访问控制、工作空间感知、并发控制、同步、异步等几个方面的研究是主要方向[44]。同时，随着计算机技术和网络技术发展，给协同软件注入了新的活动力，使其协同软件功能更具有集成性、分布性和可用性。特别地，将电子商务、商业智能、云计算模式引入协同软件中，可以进一步增强协同软件的协同性和落地性。图2-56是引入新技术的协同软件体系结构。在图中：

1．用户层

用户层由公共门户、客户门户、供应商门户、管理门户、员工门户等五大群体门户组成，每个群体操作的业务功能和实现的需求是不同的，但都可以集成在一个Web窗体上实现单点登录。

2．企业内部层

主要是将企业内部的业务功能集成在一起进行管理，其目的是企业内部的业务功能只能对内部人员使用和访问，这样可以增加企业内部的信息的安全性。这些功能包括项目流程、文件收发、费用报销、公务出差、公共资产管理、财务管理等。

3．企业外部层

企业外部是与企业产生经济效益的各类业务系统，包括产品商务、销售管理、物流管理、采购管理、库存管理、智能决策、流程管理等。

4．协同模式层

协同模式就是实现协同软件构架的模式，传统协同软件构架模式与传统的软件构架模式很相似，不同之处就是在于协同软件引入协同的概念和方法。图2-56所示是在传统构架的基础上重新引入企业集成、电子商务、电子政务、商业智能、遗留系统集成等模式，从而丰富了协同模式的内容。

5．协同技术层

协同技术就是实现协同软件所采用的技术，传统的一般是采用分布式计算、网络计算、普适计算、P2P计算和多代理系统技术及相关的技术来实现[44]，图2-56中所示主要强化SOA、企业服务总线和云计算在协同技术中的应用。

6．协同组件层

为了提高协同软件的开发效率和可伸缩计算的能力，以及协同软件的可复用性，采用组件化实现协同构架和研发是一个满足各需求的有效选择。协同组件层包括：云计算中的IaaS、PaaS、SaaS、构件和其他相关的分布组件。

7．群组通信

群组通信技术一般是指基于TCP协议的和基于UDP协议的两种通信方式，但在基于Web的群组通信是TCP和UDP更上一层的HTTP、SOAP等上实现。

图 2-56　新型协同软件层次体系结构

2.4.3　协同软件模式

协同构件模式通常情况下是指采用什么的软件构架模式来构架协同软件，并用什么技术来实现。特别是采用 SOA 模式来构架协同软件是当前的主流技术，因此，以下从协同软件构架模式和 SOA 与软件构架两个方面进行概述。

2.4.3.1　协同软件构架模式

对于不同的协同需求，需要采用不同的构架模式来指导协同软件构架和研发，下面选择面向服务、云计算、电子商务、商业智能四种模式进行概述协同软件的构架模式。

1．面向服务的协同软件构架模式

服务计算是以服务为基本单元、按需求动态变化所构建的一种松散耦合、高粒度、可伸缩的计算模式，支持分布式应用的快速、低成本的组合式开发；服务是自治的、平台独立的计算实体，支持以平台无关的方式进行使用；并通过 Web 服务来实现服务需求动态交互，在这一过程中通过 WSDL、UDDI 来完成，即通过服务的描述、发布、发现和动态组合能够开发分布式的、支持互操作和动态演化的应用系统。这是因为服务计算具有以下优点：

（1）良好的封装性。Web 服务既然是一种部署在 Web 上的对象，自然具备对象的良好封装性，因此对于消费者而言，用户能且仅能看到该对象提供的功能列表。

（2）标准协议性。这一特征从对象而来，但相比一般对象，其接口规范更加规范化和易

于机器理解。首先，作为 Web 服务，对象界面所提供的功能应当使用标准的描述语言来描述（比如 WSDL）。其次，由标准描述语言描述的服务界面应当是能够被发现的，因此这一描述文档需要被存储在私有的或公共的注册库里面。

（3）松散耦合性。这一特征也源于对象/组件技术。当一个 Web 服务的实现发生变更的时候，调用者是不会感到这一点的。对于调用者来说，只要 Web 服务的调用接口不变，Web 服务实现的任何变更对他们来说都是透明的。

（4）高度集成性。由于 Web 服务采取简单的、易理解的标准 Web 协议作为组件接口描述和协同描述规范，完全屏蔽了不同软件平台的差异，无论是 CORBA、DCOM 还是 EJB 都可以通过这一种标准的协议进行互操作，实现了在当前环境下最好的可集成性。

Web 服务技术的发展进一步推动了服务计算在工业界中的应用，其应用的基本结构就是面向服务体系结构（SOA）。SOA 具备服务自治、松散耦合、面向重用、基于契约的特性，支持面向业务的应用封装，但 SOA 实现主要技术仍是 Web 服务技术，还有以 SOA 为基础的 SCA（Service Component Architecture）、SDO（Service Data Objects）构件模式。其中：

（1）SCA：SCA 把业务流程从业务逻辑中分离出来，从而实现简化 SOA 构建的业务应该程序的创建和集成，且提供了构建由细粒度到粗粒度组装的组件机制，如图 2-57 所示。并且 SCA 还有如下优势：

① 简化业务组件开发；

② 简化作为服务网络构件的业务解决方案的组装部署；

③ 提高可移植性、可复用性和灵活性；

④ 通过屏蔽底层技术变更来保护业务逻辑资产；

⑤ 满足 SDO 数据访问；

⑥ 提高可测试性。

图 2-57　服务组件体系结构

SCA 在构建面向服务的实体时，以实现提供服务和其他服务的组件和组装组件。在具体实施时，服务实现是业务逻辑的具体实现。

（2）SDO：SDO 是一种数据编程体系结构和 API（相关 SDO 的详细叙述见 5.6.4.2 小节），可对异类数据源的统一数据访问，且提供多种不同种类数据的公共方法。目前的 SDO2.1 规范描述了静态和动态 API 创建 SDO 的两种方法，如图 2-58 所示，主要表现在：

① 简化 J2EE 数据编辑模型；

② 抽象 SOA 中的数据；

③ 统一数据应用程序的开发；

④ 支持和集成 XML；

⑤ 提供统一的数据访问框架。

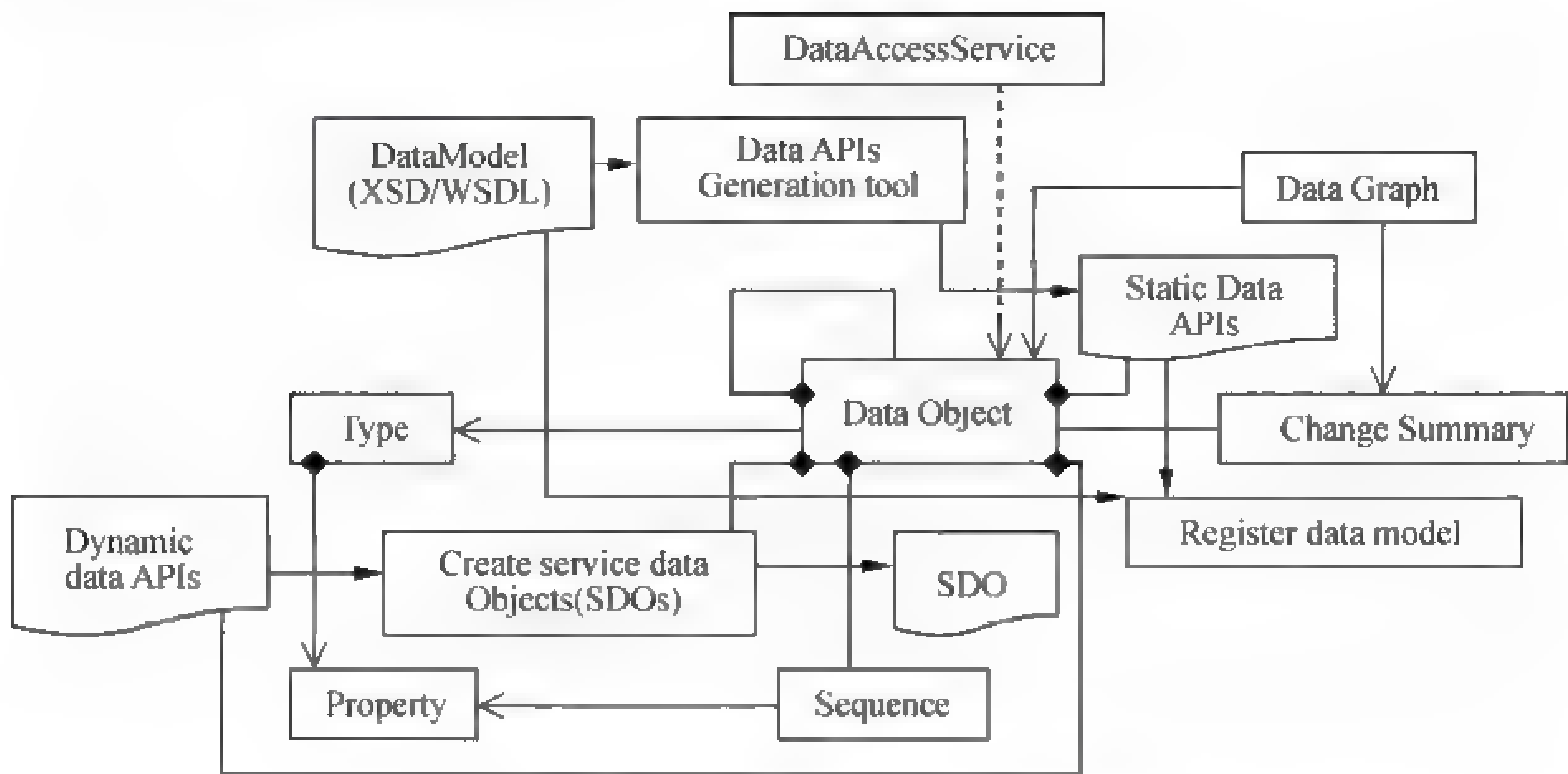

图 2-58 SDO、静态与动态 API 用法

SDO 由 SDO 编程模型和 API、Data 中介服务、数据源、数据对象、数据图、变更摘要、属性、类型和序列组成。表 2-20 所示是 SDO 在不同数据源中的应用方法。

表 2-20 SDO 在不同数据源中的应用方法

应用程序体系结构的领域	SDO 使用方法
SOA	SDO 是服务的输入和输出
数据访问	SDO 访问关系型，XML，EJB，JDO 和 Hibernate，iBATIS 数据源.SDO 是 DTO
Web 服务	SDO 表示网络上的 XML
消息传递	SDO 表示消息
XML	使用 SDO 的情况：支持 XML 的应用程序。访问 XML 文件、文档、资源和消息
连接器/适配器（EIS/CICS）	SDO 表示数据记录
EJB	SDO 是 DTO（也被称作值对象）；J2EE 设计模式
ADO.NET	ADO DataSet 是 SDO 数据图的子集
企业服务总线（ESB）	SDO 是服务的输入和输出
跨语言编程模型	完整的应用程序可能横跨层和语言。用于很多种语言技能集的相同的编程模型
模型驱动的体系结构（MDA）	SDO 模型（类型（Type）和属性（Property））是通过 UML 类和组件定义的 SDO 应用程序遵循 UML 序列（Sequence），流（Flow），状态（State）和协作（Collaboration）
Java	SDO 是带有 POJO 接口的智能的 POJO

2．基于云计算的协同软件构架模式

可以说，云计算的发展，并与服务计算相对应，可以进一步提高网络计算能力，增加软件研发效率，也更能进一步为所谓的网构软件、网络化软件动态演化服务，也可以提供可操作的 PaaS、SaaS 生成环境，从而按需求自动生成软件。这是因为云计算一般情况下，有如下

特点：

（1）安全性。缩短单机密集数据处理任务时间，把处理任务分配到各个结点计算，提高了效率。但用户关注传输到云计算端的敏感处理数据是否安全。

（2）可靠性。减少用户购买物理硬件设备的费用，资源以服务的方式进行租赁，降低用户资金投入的前期风险，促进用户把精力投入业务中。虽然用户不需要维护软件、硬件。但是用户使用云计算服务的质量依赖云计算本身的质量。

（3）可维护性。提供专业的软件管理和维护服务，减少了普通用户软件平台的日常维护管理成本。但是否所有的软件应用都适合在云计算环境下开发应用，而以往的软件应用如何移植到云计算环境下。

（4）交互性。用户可以根据业务需要动态地按需请求云计算服务，处理高峰期负载并在非高峰期释放资源。但云计算服务提供商的实际扩展能力有限，需要多个云计算服务商间的交互，而云计算服务之间的交互性较差。

同时，云计算还具有其他模式没有的或更优化的特征：① 基于虚拟化技术快速部署资源或获得服务；② 实现动态的、可伸缩的扩展；③ 按需求提供资源、按使用量付费；④ 通过互联网提供、面向海量信息处理；⑤ 用户可以方便地参与；⑥ 形态灵活，聚散自如；⑦ 减少用户终端的处理负担；⑧ 降低了用户对于IT专业知识的依赖。

3．面向电子商务的协同软件构架模式

电子商务（Electronic Commerce）是指利用计算机技术、网络技术和远程通信技术，实现整个商务（买卖）过程中的电子化、数字化和网络化，并将与消费及相关的产业采用电子商务模式实现网上消费，即通过网络将企业产品及相关的商务产品实现网上交易，并且促使电子商务具备商务性、服务性、集成性、可扩展性、安全性和协调性。

根据不同的需求将协同软件采用电子商务进行构架是一个有效的方式，这是因为电子商务不但具有采用电子的模式实现消费，还具有以下优点：

（1）能大大提高了通信速度和定制的效率，尤其是区域范围内的通信速度，能为商务产品快速获取与商务产品相关的信息提供了帮助。

（2）能节省企业的潜在开支，为企业增加收入。

（3）增加了消费者和企业间的联系。

（4）提高了服务质量，能以一种快捷方便的方式提供企业及其相关产业信息及消费者所需的服务。

（5）提供了交互式的销售渠道。使企业能及时得到市场反馈，改进本身的服务工作。

（6）可以为旅客提供全天候的服务，即每年365天，每天24h的服务。

（7）能增强旅游地间的竞争力和系统的协同计算能力。

4．面向商业智能的协同软件构架模式

商业智能（Business Intelligence），又称商业智慧或商务智能，指用现代数据仓库技术、线上分析处理技术、数据挖掘和数据展现技术进行数据分析以实现商业价值。并且商业智能作为一个工具，是用来处理企业中现有数据，并将其转换成知识、分析和结论，辅助业务或者决策者做出正确且明智的决定。是帮助企业更好地利用数据提高决策质量的技术，包含了从数据仓库到分析型系统等。而旅游商业智能则是采用这些技术来实现旅游商业价值。图2-59是一个商业智能流程处理图。

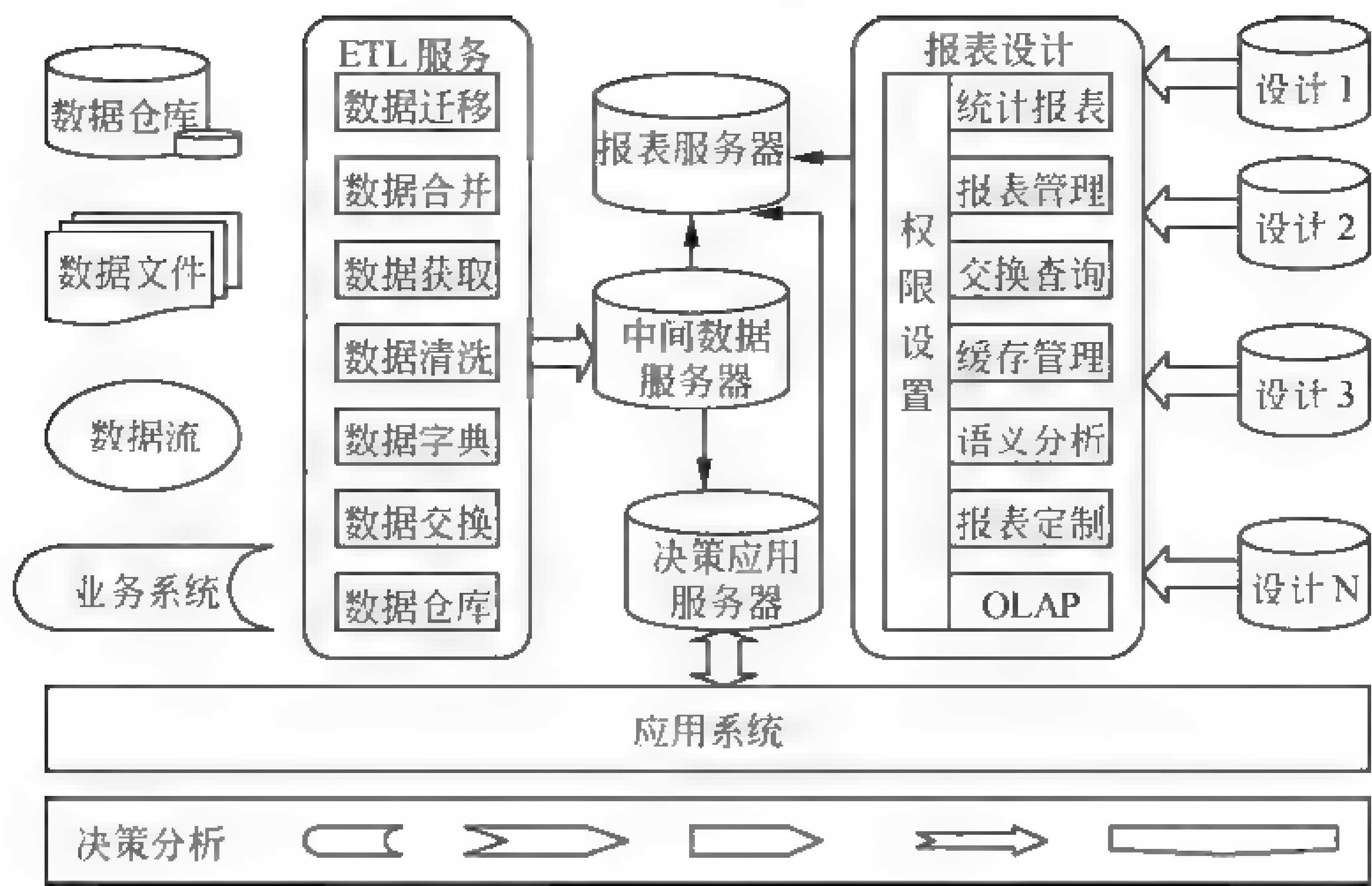

图 2-59　商业智能处理流程图

2.4.3.2　SOA 与其他协同软件构架模式的有关系

随着 IT 技术的发展，SOA 和企业架构（Enterprise Architechture，EA）逐步融合，并且成为电子商务和商业智能实现的一种方法和技术支撑，从而使其形成了新的架构理论和新的解决方案。下面从 SOA 与 EA、电子商务（EC）、商业智能（BI）和遗留系统等几方面概述服务计算的应用模式。

1．SOA 与 EI

企业架构（Enterprise Architecture，EA）的概念产生于 1987 年，在 IBM 的一个内部刊物上发表的一篇文章“A Framework for Information Systems Architecture”[作者 J.A. Zachman（扎克曼）]中提出。概念的提出是为了应对日益复杂的 IT 系统，以及高投资、低回报的问题。他认为使用一个逻辑的企业构造蓝图（即一个架构）来定义和控制企业系统及其组件的集成是非常有用的。为此，Zachman 开发了信息、流程、网络、人员、时间、基本原理等六个视角来分析企业，也提供了与这些视角相对应的六个模型，包括语义、概念、逻辑、物理、组件和功能等模型。随着 EA 的发展，产生了很多的流派，当前主要的 EA 架构包含：通用框架 Zachman、TOGAF（The Open Group Architecture Framework），以及适用于政府和军方的美国联邦政府的标准架构 FEA、美国国防部的 DoDAF 等。这些模型主要分成两派，如今正在逐步的融合在一起。随着企业架构的不断进化，企业架构理论越来越与战略和业务相融合，逐步形成了企业战略、业务架构、IT 战略、IT 架构等四个层次的 IT 规划方法论。IT 架构包含数据架构、应用架构、技术架构和 IT 治理等四个方面的内容，其中技术架构包含集成平台、公共服务平台、基础平台（软件和硬件）和安全平台等，如图 2-60 所示：

2．SOA 与电子商务（EC）

目前 SOA 通过 IBM 定义的五个切入点来实现预定义的 SOA 解决方案，就包括实现动态、松耦、伸缩的电子商务平台，从而从中获益。这些切入点同时受到业务需求（人员、流程和信息切入点）和 IT 需求（连接性和重用切入点）的驱动。基于此，IBM 技术专家在参考实际的客户经验和多年的积累的情况下，认识到业务部门在设计和实现 SOA 解决方案的

过程中经常会遵循多个常见的场景。通过定义这些场景，IBM 提供了预定义的真实方法，帮助实现 SOA 解决方案。每个场景都提供了经过测试和集成的产品或实现，用于实现此场景。因此，可以将这些场景映射到区域旅游电子商务具体的目标和需求，从而更好的实现这些系统。

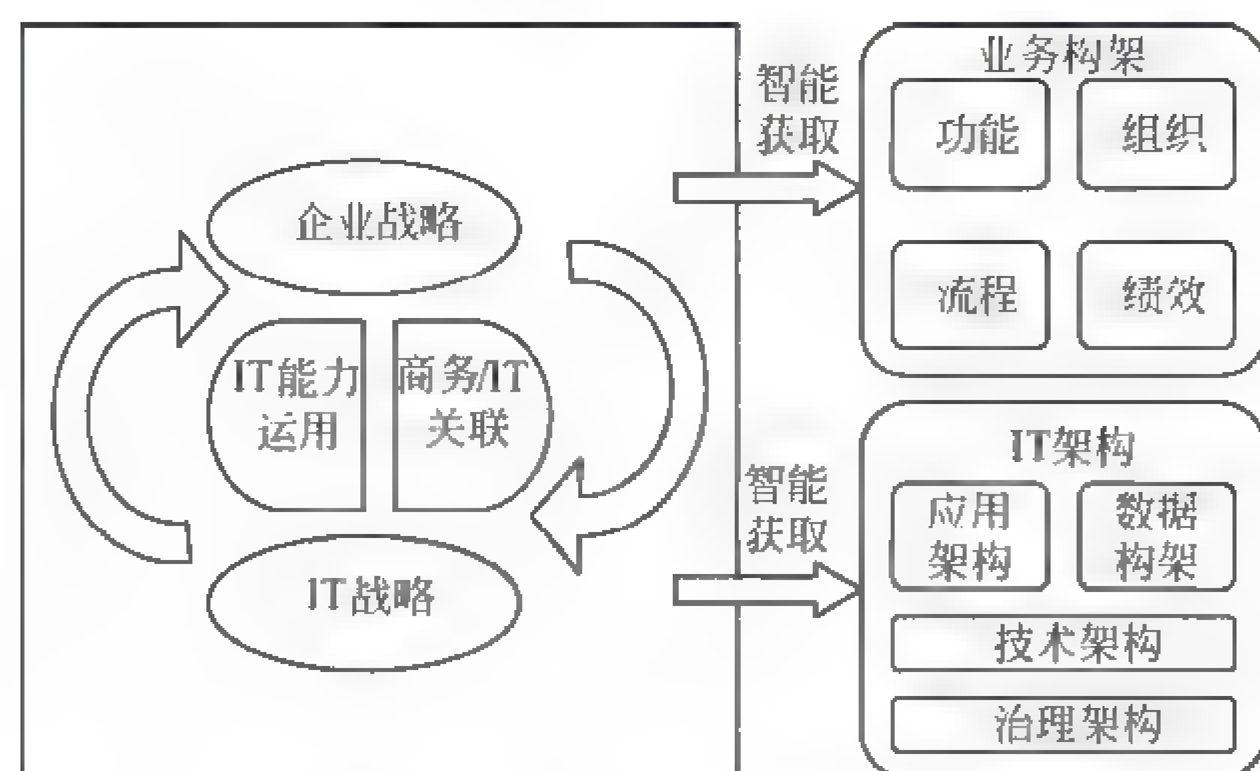

图 2-60　SOA 与 EA 集成结构

然而 SOA 是从企业的需求开始，把 IT 系统和商业流程连合在一起，以服务集成形式实现新的而灵活的应用功能。并且 SOA 简化了 IT，让 IT 变得更有弹性，以便更好地发展和优化业务流程。从而旅游企业与合作伙伴的业务需要，也使供应商与客户之间动作流程的端到端整合，让旅游企业可以快速灵敏地呼应客户和市场不断变化的需求，实现随需求应变企业。但在一个商务过程是以一套特定顺序被调用来实现某个商务目标的商务活动流。商务过程定义了流的顺序、如何处理外部事件、如何与人交互和条件判断。如图 2-61 所示：

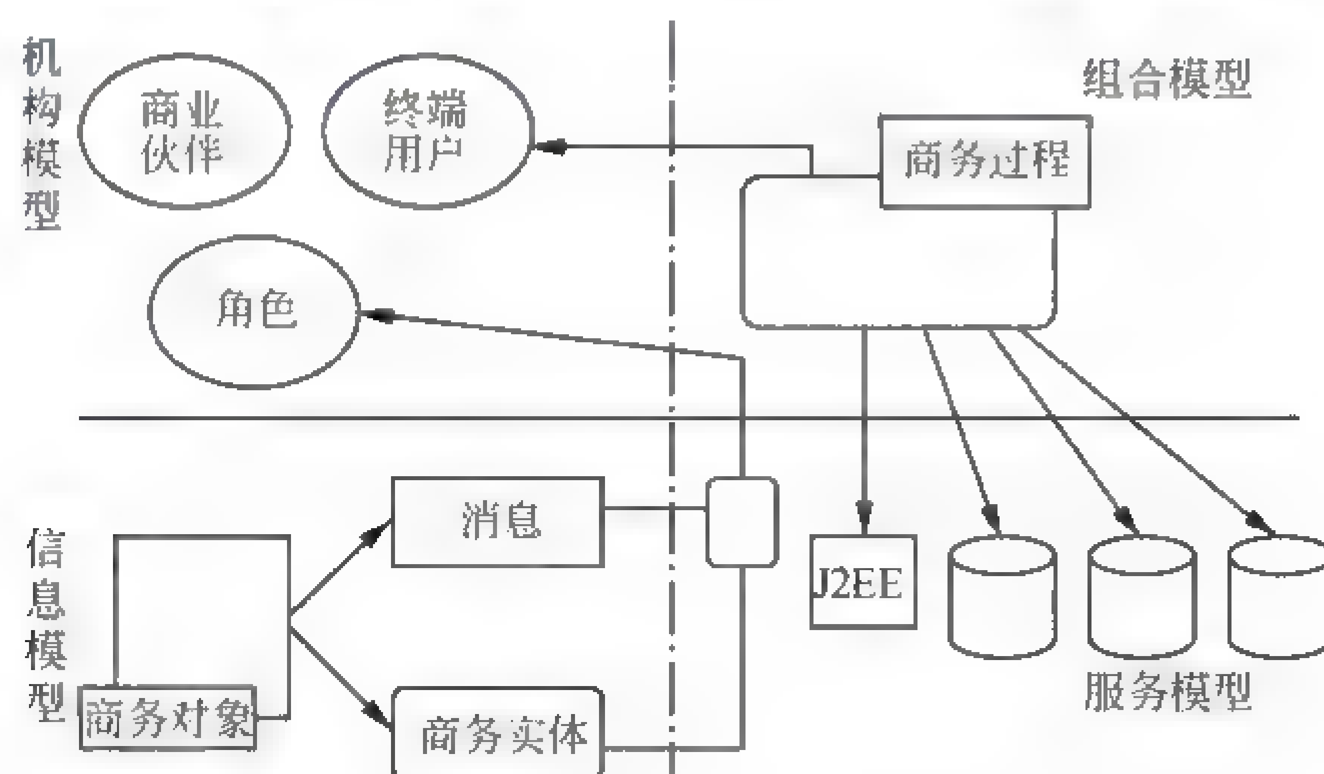

图 2-61　面向 SOA 的协同电子商务模式

3. SOA 与商业智能（BI）

SOA 是一种架构 IT 系统的方法，它将应用和 IT 功能划分为单独的业务功能和模块，即所谓的服务。用户可以构建、部署和整合这些服务，且无需依赖应用程序及其技术平台，从而提高应用的灵活性。这种业务灵活性可使企业和机构加快发展速度，降低总体拥有成本，及时、准确地获取信息，同时有助于实现更多的资产重用。而建设 SOA 体系架构就需要建立一个一致的架构框架，在这种框架中，可以快速地进行开发、集成和重用应用系统。而对

于原有的应用系统来说，可以采用合适的技术手段进行平滑的优化与过渡。而 SOA 与 BI 集成主要包括 BI 的以下几门方法与技术。图 2-62 是 SOA 与商业智能的关系图。

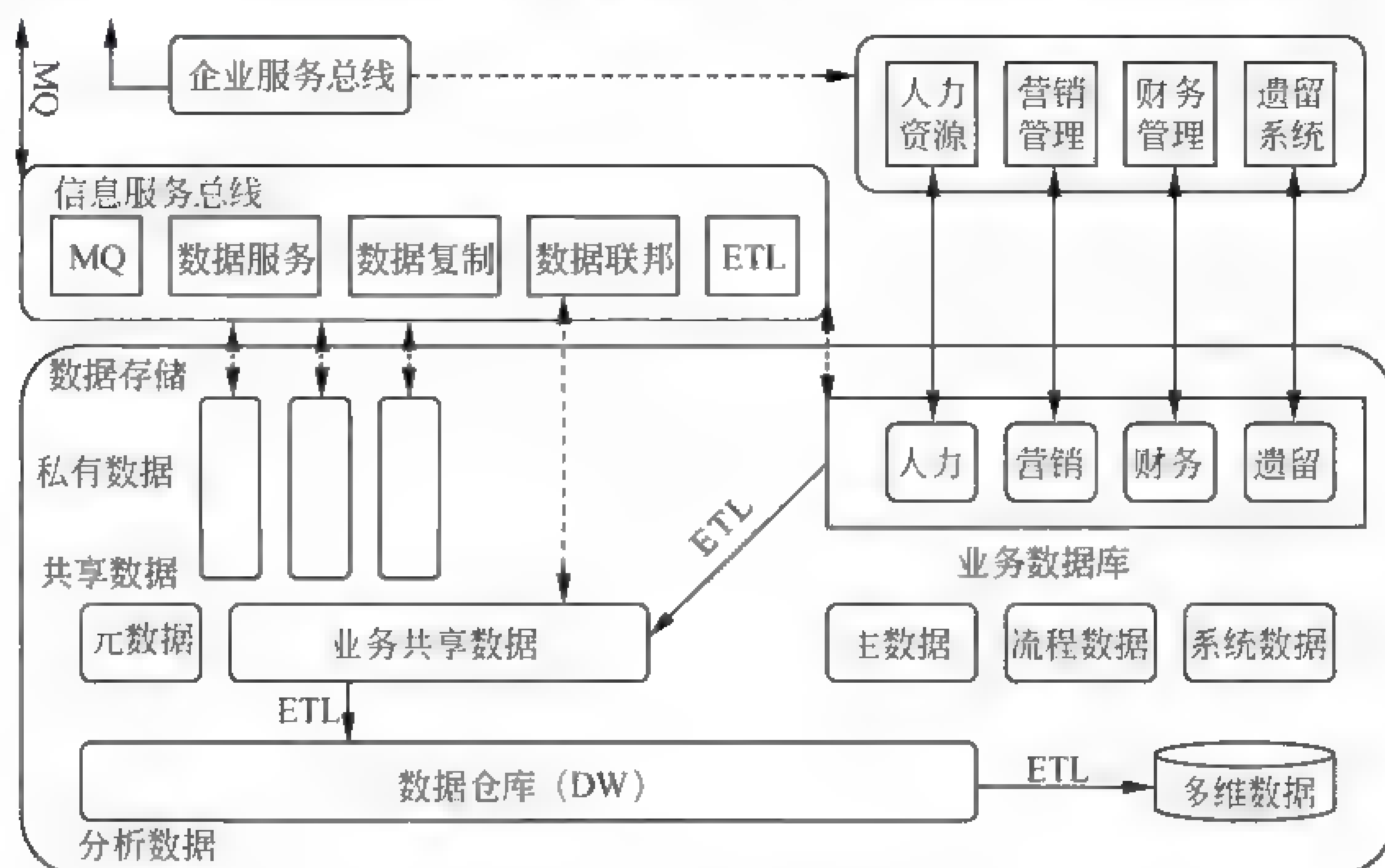

图 2-62　SOA 与商业智能的关系图

（1）数据仓库。数据仓库（Data Warehouse，DW）是一个面向主题、集成、时变、非易失的数据集合，是支持管理部门的决策过程（W.H.Inmon）。数据仓库具备以下四个关键特征：面向主题（Subject Oriented）的数据集合；集成（Integrated）的数据集合；时变（Time Variant）的数据集合；非易失（Nonvolatile）的数据集合。根据数据仓库所管理的数据类型和它们所解决的企业问题范围，一般可将数据仓库分为下列四种类型：主数据、操作型数据库（ODS）、数据仓库（DW）和数据集市（DM）。其中：

① 主数据。主数据（Master Data，MD）指系统间共享数据（例如，客户、供应商、账户和组织部门相关数据）与记录业务的活动、变动较大的交易数据相比，主数据（又称基准数据）变化缓慢，一般每年的变化在 20%左右。在正规的关系数据模型中，交易记录（例如，订单）可通过关键字（例如，订单或发票编号和产品代码）调出主数据。有关主数据管理实施的复杂程度，参照 Jill Dyche, Evan Levy 的观点大体可以把主数据管理可以分为五个层次，其中 Level 3（通过集中的总线处理，类似于翻译器）可以实现企业内任意两个系统交换数据。Level 3 是将数据转换逻辑集中化和标准化，它支持主参照数据的分布式存在（即分布的主数据存储，集中而标准的主数据转换），Level 3 打破了各个独立应用的组织边界，使用各个系统都能接受的数据标准统一建立和维护主数据（MDM）。而最高级别 Level 5（企业数据集中），当主数据记录的详细资料被修改后，所有应用的相关数据元素都将被更新，本级别可以通过 SOA 的架构平台实现。

② 操作型数据库（ODS）。ODS 既可以被用来针对工作数据做决策支持，又可用做将数据加载到数据仓库时的过渡区域。与 DW 相比较，ODS 有下列特点：ODS 是面向主题和面向综合的；ODS 是易变的；ODS 仅仅含有目前的、详细的数据，不含有累计的、历史性的数据。

③ 数据仓库（DW）。通用数据仓库，它既含有大量详细的数据，也含有大量累赘的或

聚集的数据，这些数据具有不易改变性和面向历史性。此种数据仓库被用来进行涵盖多种企业领域上的战略或战术上的决策。

④ 数据集市（DM）。是数据仓库的一种具体化，它可以包含轻度累计、历史的部门数据，适合特定企业中某个部门的需要。

（2）ETL。ETL 是 Data Extraction, Transformation and Loading 的首字母缩写，是数据仓库、数据挖掘以及商业智能等技术的基石，其主要用来实现异构多数据源的数据集成。是作为 BI/DW（Business Intelligence）的核心和灵魂，能够按照统一的规则集成并提高数据的价值，是负责完成数据从数据源向目标数据仓库转化的过程，是实施数据仓库的重要步骤。在整个项目中最难部分是用户需求分析和模型设计，而 ETL 规则设计和实施则是工作量最大的，约占整个项目的 60%～80%，这是国内外从众多实践中得到的普遍共识。同时，ETL 是数据抽取（Extract）、转换（Transform）、清洗（Cleansing）、装载（Load）的过程。是构建数据仓库的重要一环。用户从数据源抽取出所需的数据，经过数据清洗，最终按照预先定义好的数据仓库模型，将数据加载到数据仓库中去，如图 2-63 所示。

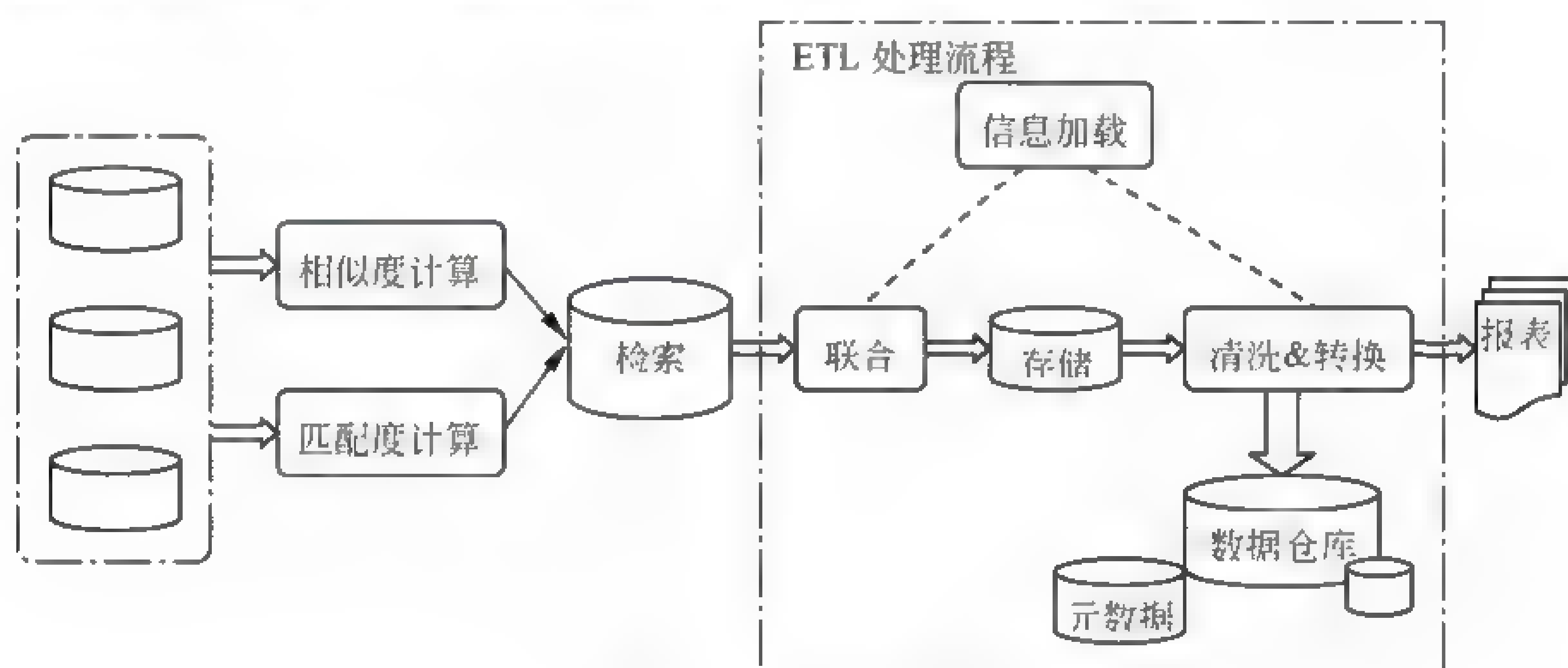

图 2-63　ETL 操作流程

（3）线上分析处理（On-Line Analytical Processing，OLAP）是一套以多维度方式分析资料，而能弹性地提供积存（Roll-up）、下钻（Drill-down）和枢纽分析（pivot）等操作，呈现整合性决策资讯的方法，多用于决策支持系统、商务智能或数据仓库。其主要的功能在于方便大规模数据分析及统计计算，对决策提供参考和支持。OLAP 需以大量历史资料为基础配合上时间点的差异，并对多维度及汇整型的资讯进行复杂的分析。OLAP 需要使用者有主观的资讯需求定义，使得系统效率较佳。

而 OLAP 的概念，在实际应用中用广义和狭义两种不同的理解。广义上的理解与字面意思相同，即针对于与 OLAP 相关的 OLTP（On-Line Transaction Processing）而言，泛指一切不对数据进行输入等事务性处理，而基于已有数据进行分析的方法，更多的情况下 OLAP 是被理解为其狭义上的含义，即与多维分析相关，且通过基于立方体（CUBE）计算而进行的分析。

（4）数据挖掘。数据挖掘是数据库知识发现（Knowledge-Discovery in Databases，KDD）中的一个步骤。数据挖掘一般是指从大量的数据中自动搜索隐藏于其中的有着特殊关系性（属于 Association rule learning）的信息的过程，它通常通过统计、在线分析处理、情报检索、机器学习、专家系统（依靠过去的经验法则）和模式识别等诸多方法来实现按需数据获取的目标。

4. SOA 与遗留系统

面向服务的体系结构（Service-Oriented Architecture）是很多业务转换工作的核心。很多企业采用增量式方法进行 SOA 转换，使用其宝贵的遗留 IT 系统作为服务提供者参与其中。但是解决方案架构师面临的挑战不仅是将 SOA 基础设施作为促进转换的手段交付，而且还要确保企业级业务操作保持可靠性和兼容性。使企业必须制定可作为 SOA 一部分的企业信息管理策略，并跨所有业务操作保持总体数据和内容的一致性。图 2-64 所示是 SOA 与遗留系统集成结构。

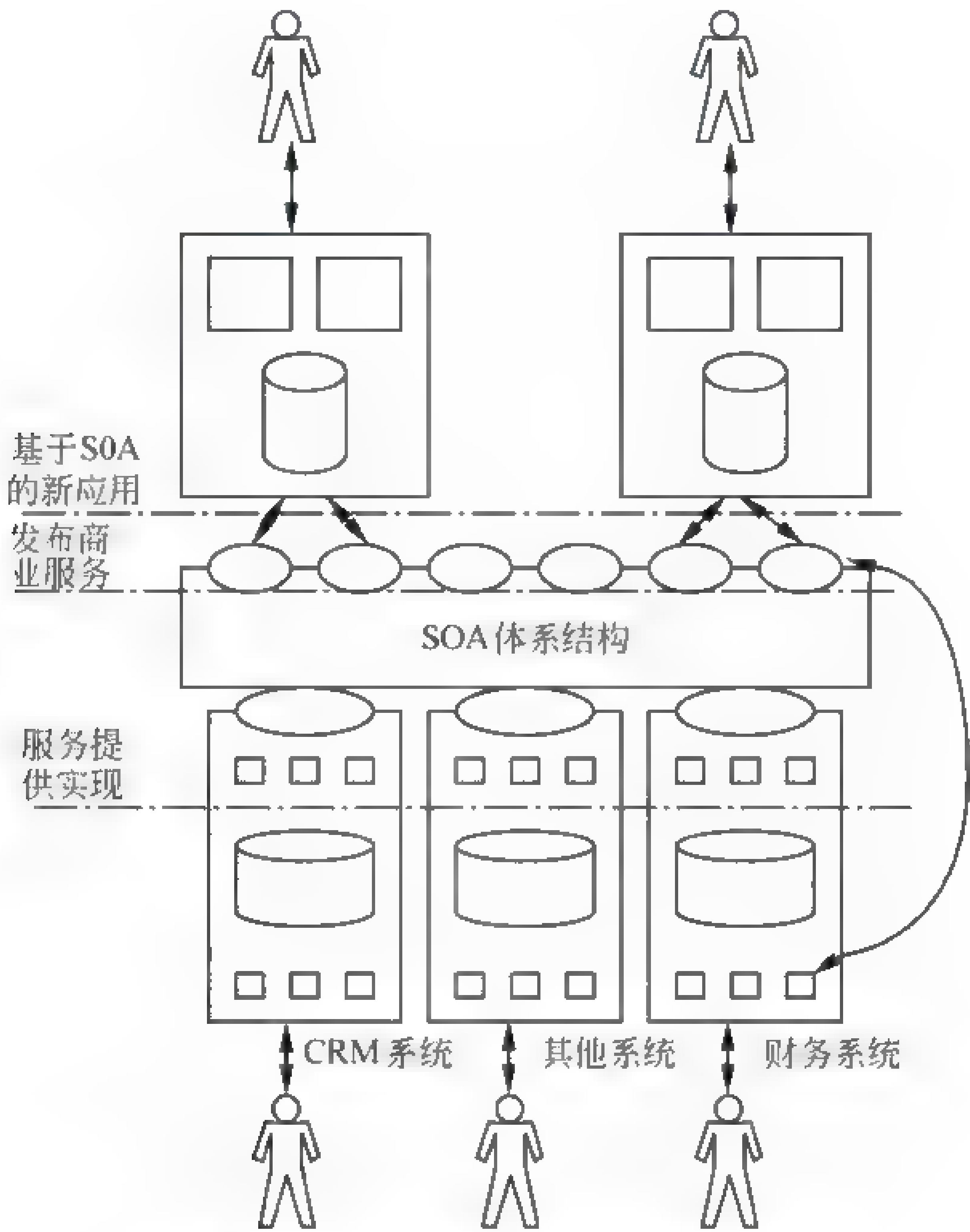

图 2-64　SOA 与遗留系统集成结构

SOA 与遗留系统集成时必须遵循以下四个基本原则：

（1）企业必须继续提供一组总体可靠的兼容业务操作与遗留系统连接、通信。

（2）需要治理流程、设计方法来确保总体业务服务模型与 IT 系统间实现全盘转换原则，使其到达数据一致性转换。

（3）企业必须实现业务与 IT 服务管理的集成视图，以使能够进行度量和交付响应变更。

（4）SOA 与遗留系统集成还需求满足引用完整性、应用程序行为相符性、跨应用程序完整性、事务处理业务逻辑一致性、数据实体处理规则完整性等原则。

2.4.4　协同软件角色

协同软件怎样实现各方群体协同，实现各方群体协作，完成同一工作，是协同软件研发

的重要组成部分。目前常见的协同软件的群体协作模型有会议模型、过程模型、活动模型、层次模型和面向对象多层次协同模型等。但这些模型在处理同步、异步协作时，显得不够灵活多样，这时人们就提出了一种基于角色的群体协作模型。根据参考文献[44]，角色协同有以下属性：

（1）协作活动的主持人可以简便地指定清晰的角色权利和角色义务使用户易于理解。

（2）协作活动的主持人可以按照清晰的角色规范评价用户的角色行为。

（3）协作活动的主持人可以灵活简便地调整角色的权利和义务。

（4）用户可以灵活简便地从一个角色转换到另一个角色。

（5）用户和协作活动主持人之间可以对角色规范进行简便的协商。

（6）所有协作者之间的交互都是通过角色进行。

因此，根据这些属性，角色协作的研究主要是根据不同的需求实现协作角色定义和分配，在不同角色协作中的角色转换，基于角色协作的协同软件构架，基于角色的协调，基于角色的界面设计，基于角色的冲突处理和基于角色的信息共享等七个方面。相关研究也从七方面及相关方面展开研究，其中研究最多的是基于RBAC（Role-Based policies Access Control）模型的协同角色控制方法，主要包括以角色为基础的权限分配的业务过程协同建模方法[45]，协同环境下基于角色的细粒度委托限制框架[46]等方案，其应用策略主要包括有基于角色协作的电子政务协同设计，面向协同装配设计的基于角色显示分析设计和企业内部基于角色协作的个性化搜索系统等。

2.4.5 协同软件的工作流技术

工作流（Workflow）是工作流程的计算模型，即将工作流程中的工作前后组织在一起的逻辑和规则，并在计算机中以恰当的模型表示和实施计算。而工作流技术来源于计算机支持协同工作领域，在最近几年已引起了普遍关注，也应用到了其他领域。工作流管理是一种能够使业务流程自动化执行的技术，是为实现工作流执行状态的实时获取，便于系统分析、决策和监控，因此被WfMC（Workflow Management Coalition）定义为工作流参考模型的重要组件之一；也是系统掌握工作流执行动态、提高可靠性和柔性不可或缺的重要功能。现代企业强调将传统的以职能为基础的组织机构和运作机制转变为以过程为中心的信息集成，以及以需求为中心的业务集成。工作流技术是实现过程集成的有效途径之一，而工作流是将应用逻辑或过程逻辑分离，并将管理知识中有关过程的知识剥离出来，使其通过信息化自动实现。

流程协同技术通常包括人与人之间的协同，系统与系统之间的协同和人与系统之间的协同。然而，WFMC将工作流定义为：工作流是一类能够完全或者部分自动执行的业务过程，它按照一系统过程规则，把文档、信息或任务在不同的执行者之间进行传递和执行。同时，还包括一组活动及相互间的顺序关系。也包括过程及活动的启动和终止条件，以及对每个活动的描述，因此，WFMC工作流的定义为流程协同提供了方法和策略。使得流程协同就是实现贯穿在各种信息结点的业务流程间的同步和异步操作，使之相互协作，实现整个业务流程。图2-65所示是工作流实现模型，图2-66所示是协同工作流执行模型。

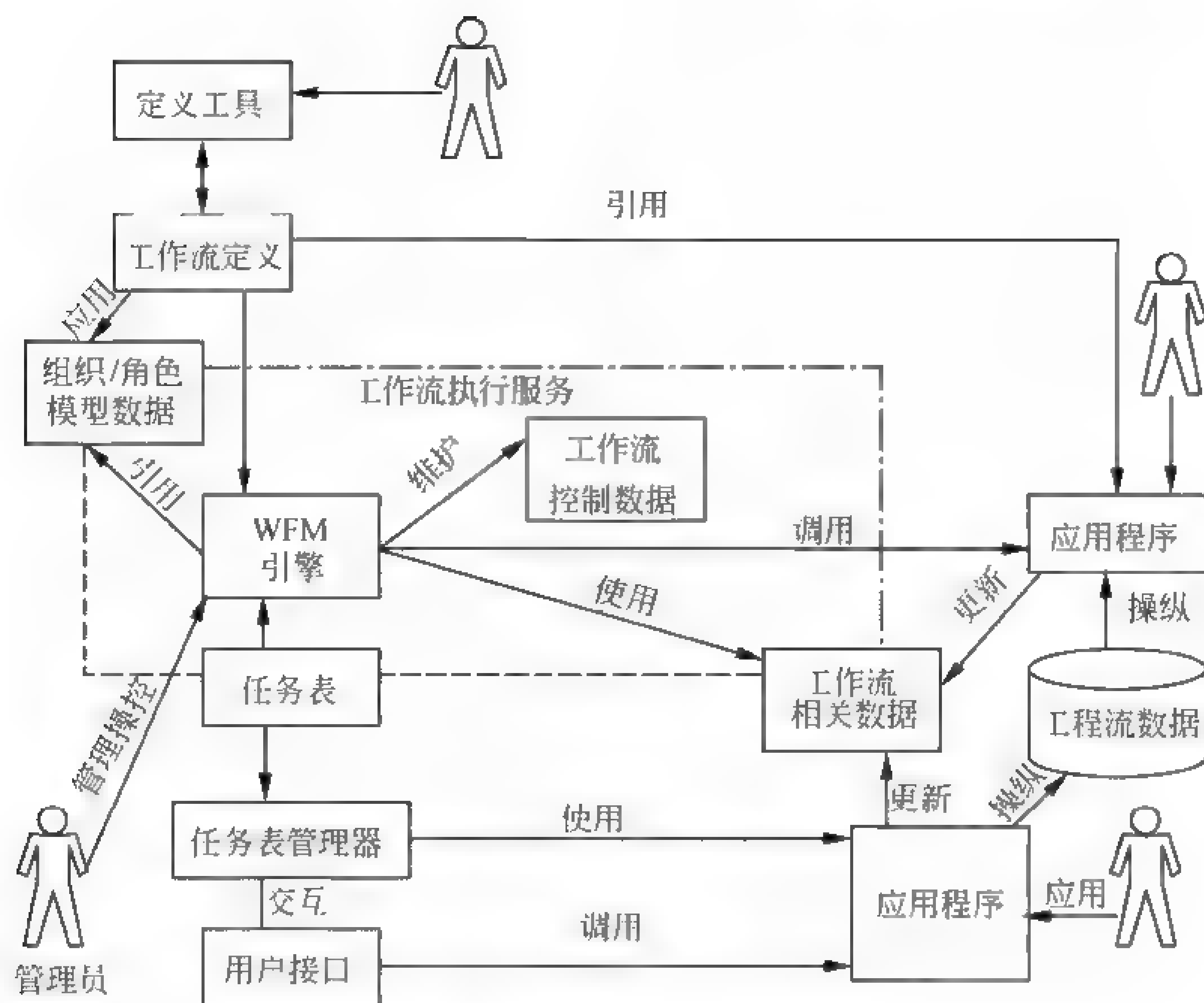

图 2-65　工作流实现模型

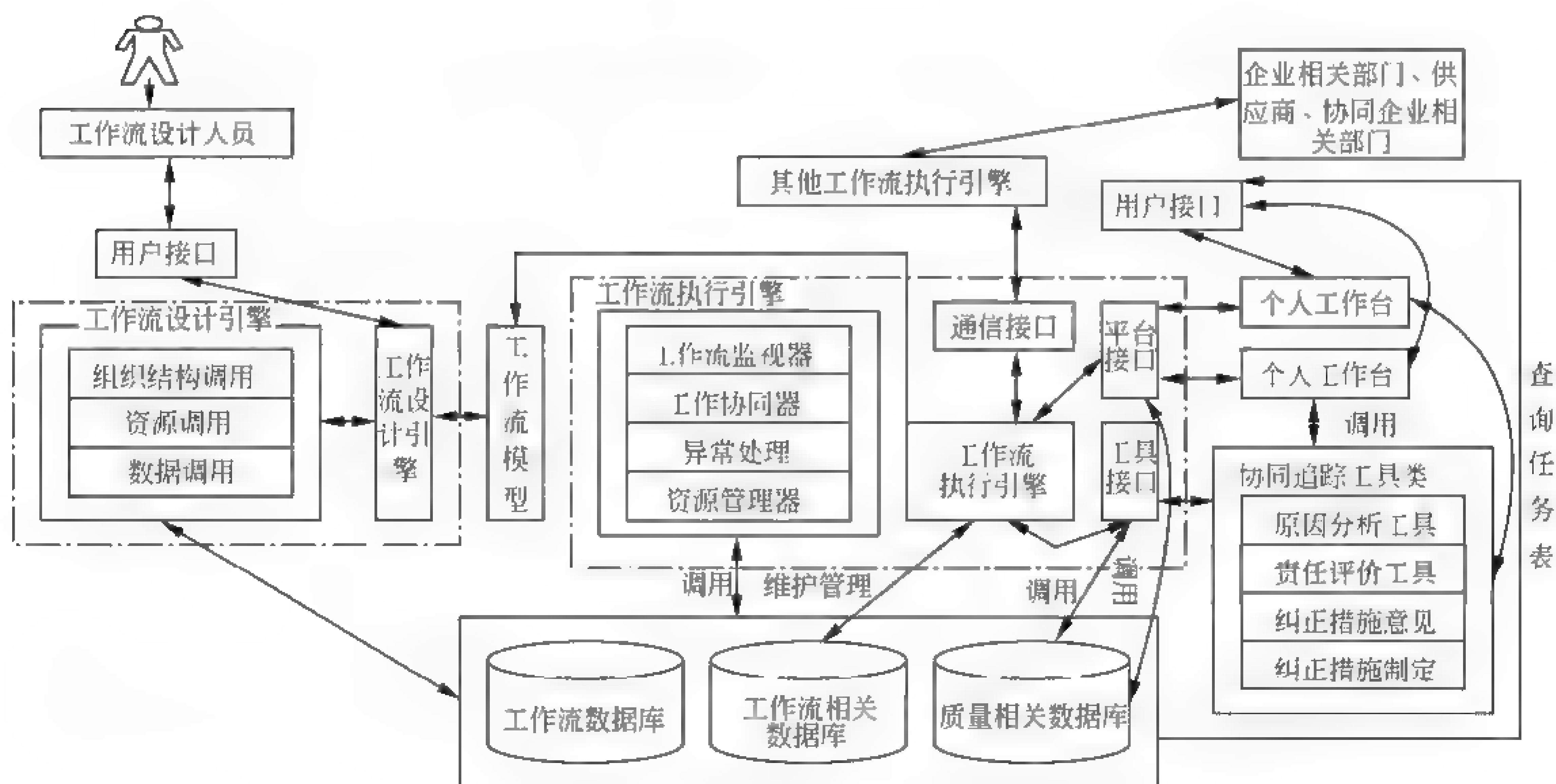

图 2-66　协同工作流执行模型

2.5　开源软件的软件开发构架模式

前面的各节已经详细概述了软件体系结构、可信软件和协同软件，以及一些其他与之相

关的软件构架方法，这些方法同样可以用来指导面向开源软件的软件开发，使其分散在 Internet 中的各开源软件能通过这些构架方法形成一种新的软件研发模式，让开源软件不但是一种高可重用、便于使用的软件实体，还是一种以开源软件为载体的新型软件研发模型。但目前的开源软件的发展仍没有强调它一种软件研发方法，仅仅是让开源软件发布出来作为一种奉献、供下载使用的一种软件呈现模式；在推动软件发展、提高软件研发效率和减小重复开发等方面给人们提高极大的便利。要使开源软件形成一种新的软件研发方法，还有许多路要走，主要表现在：

（1）怎样将众多的开源软件进行有效的分类，并指出这些开源软件的可用程度如何。

（2）怎样将众多开源软件进行评估聚集，采用 SaaS 模式进行发布给用户使用，并指出所发布的开源软件应用领域。

（3）怎样将众多开源软件集成在一个可操作的平台上，使其以服务、云的模式将开源软件提出用户；等等。

这些挑战的存在，给开源软件广泛的应用带了极大的问题。因此，本章各节详细综述了软件构架的原理、方法和相关技术，其目的就是通过这些构架方法来促使开源软件进一步发展，使其可用性更强，更能集成更多智慧。图 2-67 是具有软件体系结构、可信和协同性的面向开源软件的结构图。

图 2-67　软件体系结构、可信和协同性的面向开源软件的结构图

用软件体系结构来指导开源软件构架的目的是使面向开源软件的软件研发具有针对性、导向性，并用于指导面向开源软件的软件研发和促进开源软件自身的发展。正如前面所述一样，软件体系结构是一种指导软件分析设计的一种构架模式。在应用开源软件研发软件系统时，用软件体系结构来指导软件开发可以提升软件的生命周期和满足用户的需求。虽然开源软件的开发语言多为 JAVA、PHP、C 等基础语言，但所开发的软件是针对某一需求而完成，因此可以认为开源软件具有像构件一样的独立性、可重用性，但开源软件难以形成像构件管理系统一样的多样集成化系统来统一管理和分配开源软件。造成这方面的原因是：

（1）开源软件是针对一个特定的需求而开发的一款软件，而构件是将某一特定相同功能的进行封装，并提供接口给需求使用。

（2）开源软件具有较强的自由性，开发群体多样，受约束小，导致部分开源软件可用性不强，难以满足特定需求；而构件则不同，构件的可用性强，有专业的技术体系支持。

（3）开源软件源代码公开，在没有违反 GPL 等协议的下情况，可以任意使用，也给软件研发人员减小了重复工作量，提高了研发效率。而构件是封装好的、不公开，是具有较高商业机密的软件实体。

（4）开源软件分散性强，同一特定的软件重复较多，而真正可应用的实际工作的开源软件不多，使得人们在选择开源软件作为自己的研发工具时存在诸多困惑，但开源软件为研发人员提供研发基础是不可非议的，从侧面说明开源软件的发展还有较长的路要走，也还有较多与开源软件相关的基础工作要做；目前开源软件还是发展的初期阶段。而对构件来说，则不一样，当前面向构件的软件开发已经形成了一套较完整的体系，不论是将构件应用于实际中，还是用于研发人员研发都逐渐趋于成熟阶段。

（5）目前开源软件应用往往需要与多款开源软件集成使用，而且不同的应用领域所选择的开源软件可能是不一样的。而构件不一样，某一构件管理系统一般只应用某一领域，如电信相关的构件系统只应用于电信行业。

同样，开源软件虽然分布于 Internet，但与 SaaS 的呈现模式是不一样的，这是因为 SaaS 是以云计算思想为载体，将所有软件服务纳入云计算系统的范畴内，进行统一管理，并能提高软件的可伸缩性应用；同时，SaaS 是以云计算为基础的一种快速、共享、重用的 IT 应用程序生成方法，又称软件云服务，且在面向用户上，SaaS 具有无硬件、无维护、无升级等多种优势。而开源软件则完全不同，可以说开源软件是相对的独立的个体，很难具备 SaaS 的特征，要说有，仅是开源软件也是分布于 Internet 中的。因此，将开源软件这个庞大的群体用 SaaS 思想进行研究、进行集成将今后开源软件的应用方向和发展方向。

同时，在发展和应用开源软件时，也不得不评估开源软件的可信性，正如前面所述一样，可信性是影响软件应用精度，是增强软件的可用度一种有效的方法，软件的可信程度直接影响软件的使用的健壮性和可靠性。因此，在实践面向开源软件的软件开发时，将可信性纳入软件分析设计中，是开源软件快速、有效应用到实际过程中不得不考虑的课题。同时，由于开源软件的特殊性、自由性、分散性也决定必须评估开源软件的可信性，这是因为，当前诸多开源软件是以团体和个人的形式进行研发，使得开源软件的质量和可信性得不到保障，也难以得到技术上的支持和帮助，这些因素决定目前开源软件只能在中小企业中得到使用，不得有效的在大型软件系统中得到应用。虽然目前有数百种开源软件运行稳定性较好、也便于实现轻量级的软件研发，但是这些仍只能用于中小型软件系统中，既使要应用到大型软件系统中，也要集成多款开源软件进行实现，即使是这样，开源软件的质量和可信性也得不到保障；因为只要软件存在一点点缺陷，就有可能导致整个软件系统的崩溃。

当然，在面向开源软件的软件研发时，把协同的思想纳入其中，也是提出研发的一种有效手段，毕竟当前许多业务类的软件系统都具有协同性，需要由人与系统、系统与系统自动及人与人间协作来完成。既然要推动面向开源软件的软件研发的发展，提倡将面向开源软件作为一种新型软件开发模式，就需要考虑开源软件的协同性、广泛性，使开源软件不再是软件呈现方式，而是使开源软件逐步发展为一种研发开发模式，将协同性的优点应用到面向开源软件的软件研发中，从而从构架模式丰富开源软件的理论体系，最终使开源软件以构件、SaaS 模式发布于 Internet 中。

综上，将软件体系结构、可信性和协同性纳入到面向研源软件的软件研发中，其优点可以丰富开源软件发展的理论体系，可以使得面向开源软件的软件设计具有可靠性和协同性，

也可以将使用较多、质量好、健壮性高的开源软件进行集成、规划并形成 PaaS 和 SaaS 模式发布于 Internet 中，从而方便用户选择和使用。不像现在的开源软件零散分布在不同的站点中，可用性不高，软件的可信性低。本章采用了大量的篇幅来论述、总结软件体系结构、可信软件和协同软件的正是这个原因；特别地，软件体系结构和可信性是当前应用和研究的热点，将这两个理论应用到面向开源软件的软件开发中，必将促进开源软件的发展，进一步提高开源软件的可用性。

小　　结

本章全面概述和总结了软件体系构架、可信性和协同性，这可以为面向开源软件的软件开发提供理论体系和指导方法，特别是为面向开源软件的开发方法提供了软件设计模式，使面向开源软件的软件开发也可以采用当前的软件构架模式进行分析设计，从而使面向开源软件的软件构架具备可靠性、安全性、可伸缩性、可定制化、可扩展性、可维护性、客户体验/可用性、市场时机/市场需求。因此，在本章首先概述和总结了软件体系结构方法，这些内容包括：

（1）软件体系结构，其内容主要包括软件构架的特点、软件构架的质量评估、软件架构“4+1”视图模型、软件构架师等，并对软件体系结构进行论述，包括目前软件体系结构研究挑战与进展、构件的状况及描述方法、软件体系结构描述方式、软件体系结构风格状况、软件体系结构方法、软件体系结构可靠性等。

（2）软件层次结构，其内容包括分层模式、面向构件的分层模式、面向服务的分层模式、面向云计算的分层模式。

（3）软件中间件构架方法，也概述和总结软件中间件的软件构架方法，这种方法也是当前应用的热点和软件的构架方法。

（4）轻量级的软件构架方法，其内容主要包括原则、过程、技术、轻量级构架的实现等。

（5）可信软件的构架方法，其内容包括可信软件概述、可信软件基本原理、可信软件构造所满足的基本条件、可信软件演化、可信软件度量、可信软件技术、可信研究进展等。

（6）协同软件构架方法，其内容包括协同软件概述、协同软件原理、协同软件模式、协同软件角色、协同软件的工作流技术等。

最后将这些构架模式应用到了面向开源软件中，在模式中提出了以 SaaS 的思想来集成分布在 Internet 中的开源软件，并将在面向开源软件的软件开发中纳入了软件体系结构、层次结构、中间件模式、可信和协同模式，这样就丰富了面向开源软件的软件开发的内容和理论体系。

参 考 文 献

[1] Len Bass, Paul Clements, Rick Kazman. Software Architecture in Practice, Second Edition. Addison Wesley/Pearson，USA：2003.

[2] IBM Software Group. Rational Unified Process for Systems Engineering. http：//public. dhe. ibm. com/software/dw/rationaledge/aug03/f rupse mc. pdf.

[3] Community Software Architecture Definitions. http：//www. sei. cmu. edu/architecture/ start/ community. cfm.
[4] IEEE 计算机协会，IEEE 推荐的软件密集型系统架构描述的实践：IEEE 1471：2000.
[5] Mary Shaw，David Garlan. 软件架构——关于一个正在形成的学科的观察. Prentice Hall，1996.
[6] Rick Kazman，Len Bass. Making Architecture Reviews Work in the Real World. IEEE Software，2002，19(1)：67-73.
[7] Rick Kazman，Philippe Kruchten，Robert L. Nord，James E. Tomayko. Integrating Software-Architecture-Centric Methods into the Rational Unified Process. Software Architecture Technology Initiative. 2004，7，http：//www. sei. cmu. edu/publications/pubweb. Html.
[8] Philippe Kruchten. Architectural Blueprints—The “4+1” View Model of Software Architecture. IEEE Software，1995，12(6)：42-50.
[9] G. Booch：Object-Oriented Analysis and Design with Applications，2nd. edition，Benjamin-Cummings Pub. Co. ，Redwood City，California，1993.
[10] Garlan & Shaw：An Introduction to Software Architecture. Advances in Software Engineering. and Knowledge Engineering，Volume I，edited by V. Ambriola and G. Tortora，World Scientific Publishing Company，New Jersey，1993.
[11] 梁军涛，蒋晓原，张海. 根据构件体系评估软件的可靠性. 应用科学学报，2009，27(3)：277-281.
[12] Sam Malek，George Edwards，Yuriy Brun，et al. An architecture-driven software mobility framework. Journal of Systems and Software，2010，83(6)：972-989.
[13] 孙昌爱，金茂忠，刘超. 软件体系结构研究综述. 软件学报，2002，31(7)：1228-1237.
[14] 梅宏，申峻嵘. 软件体系结构研究进展. 软件学报，2006，17(6)：1257−1275.
[15] 黄柳青，王满红. 构件中间：面向构件的方法与实践. 清华大学出版社，2006.
[16] 梅宏，陈锋，冯耀东，等. ABC：基于体系结构、面向构件的软件开发方法. 软件学报，14(4)：721-732.
[17] 马亮，谢冰，杨芙清. 多构件库统一刻面检索机制. 电子学报，2002，12A(12)：2149-2152.
[18] 田丽从，张莉，周伯生. 软件体系结构描述语言研究现状分析. 计算机科学，2005，32(2)：109-113.
[19] PERRY D E，WOLF A L. Foundat ion s for the study of software architecture. ACM S IG SO FT So ftw a re E ng inee ring No tes，1992，17 (4)：40-52.
[20] BUSCHMANN F，MEUNIER R，ROHNERT H，et al. Pattern-oriented software architecture，volume 1 ：a systemof patterns. Chichester：Wiley，1996：229-237.
[21] SHAW M，GARLAN D. Software architecture：perspectives on an emerging. New Jersey：rentice Hall，1996 ：19-32.
[22] SHAW M，CLEMENTS P. A field guide to boxology：preliminary classification of architectural styles for software systems / /Proc of the 21st International Computer

Software and Applications Conference. Washington DC：IEEE Computer Society，1997 ：6 -13.

[23] 丁博，王怀民，史殿习. 一种面向普适计算的适应性软件体系结构风格. 软件学报，2009，20(Suppl.)：113–122.

[24] 崔晓峰，孙艳春，梅宏. 以决策为中心的软件体系结构设计方法. 软件学报，2010，21(6)：1196–1207.

[25] 余萍，马晓星，吕建，陶先平. 一种面向动态软件体系结构的在线演化方法. 软件学报，2006，17(6)：1360–1371.

[26] 黄双喜，范玉顺，赵彧. 一类通用的适应性软件体系结构风格研究. 软件学报，2006，17(6)：1338–1348.

[27] 祝义，黄志球，曹子宁，周航，刘亚萍. 一种基于形式化规约生成软件体系结构模型的方法. 软件学报，2010，21(11)：2738–2751.

[28] 朱春国，曾国荪. 一种面向方面软件体系结构模型. 计算机应用研究，2010，27(9)：3387-3389.

[29] 梁军涛，蒋晓原，张海. 根据软件体系结构评估软件可靠性. 应用科学学报，2009，27(3)：277-281.

[30] 刘克，单志广，王戟，等. “可信软件基础研究”重大研究计划综述. 中国科学基金，2008，22(3)：145-151.

[31] 沈昌祥，张焕国，王怀民，等. 可信计算的研究与发展. 中国科学：信息科学，2010，40(2)：139-166.

[32] 杨洁，杨育，王小磊. 面向可信软件的风险管理模型研究. 计算机应用研究，2008，25(10)：3010-3012.

[33] 曾晋，孙海龙，刘旭东，等. 基于服务组合的可信软件动态演化机制. 软件学报，2010，21(2)：261-276.

[34] 丁博，王怀民，史殿习，等. 一种支持软件可信演化的构件模型. 软件学报，2011，22(1)：17–27.

[35] 李季，刘春梅. 基于本体的可信软件演化框架模型. 计算机应用研究，2010，27(12)：4551-4554.

[36] Malaiya Y K，Srimani P K. Software reliability models：Theoretical developments，evaluation and applications. Los Alamitos，CA，USA：IEEE Computer Society Press，1991.

[37] Ohba M. Software reliability analysis models. IBM J Research and Development，1984，28(4)：428-443.

[38] 汤永新，刘增良. 软件可信性度量模型研究进展. 计算机工程与应用，2010，46(27)：12-16.

[39] Algirdas A，Laprie J C，Brian R，et al. Basic concepts and taxonomy of dependable and secure computing[J]. IEEE Trans Dependable Secure，2004，1(1)：11-33.

[40] 王怀民，唐扬斌，尹刚，等. 互联网软件的可信机理. 中国科学：E 辑，2006，36(10)：1156-1169.

[41] 文静，王怀民，应时，等. 支持运行监控的可信软件体系结构设计方法. 计算机学报，2010，33(12)：2322-2333.
[42] 王晓暾，熊伟. 基于 QFD 和 TRIZ 的可信软件技术冲突解决方法. 航空学院，2011，32(1)：128-136.
[43] 王怀民，唐扬斌，尹刚，等. 互联网软件的可信机理. 中国科学 E 辑，信息科学，2006，36(10)：1156-1169.
[44] 汤庸等，协同软件技术及应用. 北京：机制工业出版社，2007.
[45] 王博，张莉. 基于角色权限的业务过程协同建模方法. 计算机工程，2009，35(13)：14-16.
[46] 刘伟，蔡嘉勇，贺也平. 协同环境下基于角色的细粒度委托限制框架. 通信学报，2008，29(1)：83-91.

第 3 章　面向开源软件的分析设计方法

在第 2 章，全面概述和总结了面向开源软件的软件开发的软件构架原理，以及相关的实现方法，在本章将继续论述面向开源软件的软件开发的分析计方法，为形成一种面向开源软件的软件开发不仅在原理构架上有理论和方法支撑，而且在分析设计上也有分析指导方法。本章主要回顾当前的面向对象、面对构件的软件分析设计方法，以及面向 UML（Unified Modeling Language）的软件建模方法；论述面向服务的软件分析设计方法和面向软件语义、形式化的软件分析设计方法。

3.1　开源软件分析设计方法概述

开源软件的兴起和发展，对提高软件的复用性、开发效率，降低软件研发成本等具有重要的意义。因此，近年来逐渐形成一种面向开源软件的软件开发方法，该方法以开源软件为软件开发的载体，通过调用开源软件所提供的开发接口，直接在基于开源软件的基础实行按需求的软件开发，这种模式目前在诸多领域实现了定程度的应用。根据 Actuate 公司举办的第四届开源调查统计显示，中国大陆的 IT 业者对完全免费的开源软件是非常热衷的。今年 Actuate 公司首次与英国，北美，法国，德国等几个国家、地区一起，在中国大陆地区也进行了面向 IT 业者的调查投票。据投票结果显示，有 80.3%参与投票的大陆公司 IT 部门均表示在商务应用中部署了开源有关的技术。72.6%的公司表示掌握程序源码对公司的业务至关重要。而在德国，只有 41.4%的公司表示持有所部署软件的源码非常重要，这一比例在北美地区则是 39.9%，法国是 36%，英国则只有 35.2%。而且在中国大陆，有 67%的公司表示在公司工作所用的软件中采用了开源软件，德国的比例则是 60.6%，英国为 42.1%，北美为 41%。

同时，根据 IT168 技术调研中心 2008 年的调查结果显示：在应用层上面，桌面系统操作接近 70%，服务器操作系统接近 60%；而数据库也超过了 50%。但相对操作系统和数据库来讲，其他类型的开源软件的应用比例都没有超过 50%。这说明，开源软件的应用和服务的价值还没有得到最广泛发现。从用途分布来看，有 26%在工作场所使用开源软件，37.5%在工作场所和个人都使用开源软件，而技术人员对开源的应用体验与传播起到了关键作用。客户来源分布上，调查显示出了有接近 70%的认为企业用户会是开源软件的客户来源，其中中小企业所占比例超过了 40%。低价优质的开源软件无疑是中小企业的最佳选择。

而且开源软件目前的发展商业模式也处于发展中，主要包括如下几种[①]：

1．基于云计算的 SaaS 模式

SaaS 是随着互联网发展和应用软件的成熟，而在 21 世纪开始兴起的一种完全创新的软件应用模式。它是一种通过 Internet 提供软件的模式，厂商将应用软件统一部署在自己的服

① http://www.cnsunrun.com/news/index/2/170

务器上，客户可以通过互联网向厂商定购所需的应用软件服务，按定购的服务多少和时间长短向厂商支付费用，并通过互联网获得厂商提供的服务。对于许多小型企业来说，SAAS 是采用先进技术的最好途径，它消除了企业购买、构建和维护基础设施和应用程序的需要。这也是大多数开源软件公司采用的模式。

2．混合发展模式

开源软件的不断升温吸引了商业软件公司和政府部门的关注。由于开源软件中有很多优秀的项目和高质量的代码，如果可以直接利用这些资源的话，对于商业公司来说就可以节省大量成本。商业软件公司在基于开源代码上开发的商业软件在用于商业用途时可灵活的选择收费或者免费。而对于个人用户来说，使用这些商业软件依然免费，当然除用于商业用途除外，同时这些厂商依然可以提供有尝的技术服务支持，这样就使得这种灵活的商业模式正越来越多的受到更多的商业软件公司的欢迎。

由调查数据显示和开源软件的发展步伐可知，开源软件已经慢慢被研发人员、企业等部门接收，也正在逐渐形成一种新的软件开发方法，可以说这种方法有别于面向构件的方法（具体区别见第 2 章）。然而怎样才能使开源软件形成一种有理论、方法和技术支撑的软件开发方法呢！即以软件工程、需求工程为的思想来指导开源软件的软件开发。

（1）明确需求，即根据用户的需求，制定需求，并根据需求选择不同的开源软件，使开源软件集成在一起形成一个可进行二次开发的软件开发环境和可直接使用的开源软件满足需求。

（2）选择适合面向开源软件的软件开发的软件构架方法，正如第 2 章所述，软件构架是软件生命周期中最低层、最基础的部分，因此，对于面向开源软件的软件开发，选择好的软件构架是重要的，也是增强开源软件的健壮性、可用性所必须考虑的。

（3）选择合理软件的分析设计方法，开源软件作为一种以软件实体为中间研发的新型软件开发方法，同样需要制定一种满足面向开源软件的软件开发的分析设计方法，根据开源软件的特征，一般采用面向服务、面向语义形式分析和软件的方法和思想来指导软件的分析设计。

（4）评估开源软件，由于开源软件的各类繁多，且同一类有多款开源软件并存，不同的开源软件所表现出来的稳定性、完整性、可用性是不同的。因此，需要对即将使用的开源软件进行评估后再选择，最终获取开源软件质量好的开源软件作为软件开发使用。

最后、测试开源软件，由于很多开源软件除 Apache 等专门的开源软件组织提供外，还有很大一部分是由个人、团队及相关企业提供，使得部分开源软件可能存在软件缺陷。因此，在使用面向开源软件的软件开发中，特别是开源软件具有重要经济价值的软件系统时，需要对所评估选择的开源软件进行测试。

以上基本概述了面向开源软件的软件开发的一些基本步骤方法，相关详细将在第 4 章中进行分析研究。

3.2　基本的软件分析设计方法

软件分析设计经历了面向机器、面向过程、面向对象、面向构件、面向方面等发展过程，

各个过程都在不同时期、不同时间段推动了软件的进步，特别是与软件工程相结合以来，有效解决了软件的可复用性和所带来的软件危机。下面根据面向开源软件的软件开发需要，主要概述面向对象和面向构件的设计方法，也概述软件建模方法 UML。

3.2.1 面向对象设计方法

可以说面向对象设计方法是继面向过程方法的一次重要的革命。当前在很多软件领域仍采用面向对象的方法来指导软件设计和编程，当然大多数开源软件也是采用面向对象的方法来设计实现的，在面向开源软件的软件开发中仍有大多数开源软件采用面向对象的方法来进行设计和程序开发的。

3.2.1.1 面向对象设计发展历史

面向对象程序设计的雏形，早在 20 世纪 50 年代后期，在用 FORTRAN 语言编写大型程序时，常出现变量名在程序不同部分发生冲突的问题。鉴于此，ALGOL 语言的设计者在 ALGOL60 中采用了以“Begin…End”为标识的程序块，使块内变量名是局部的，以避免它们与程序中块外的同名变量相冲突。这是编程语言中首次提供封装（保护）的尝试。此后程序块结构广泛用于高级语言如 Pascal 、Ada、C 语言之中。而且在 1960 年的 Simula 语言中即可发现，当时的程序设计领域正面临着一种危机：在软硬件环境逐渐复杂的情况下，软件如何得到良好的维护？面向对象程序设计在某种程度上通过强调可重复性解决了这一问题。20 世纪 70 年代的 Smalltalk 语言在面向对象方面堪称经典——以至于 30 年后的今天依然将这一语言视为面向对象语言的基础。并且在 60 年代中后期，Simula 语言在 ALGOL 基础上研制开发，它将 ALGOL 的块结构概念向前发展一步，提出了对象的概念，并使用了类，也支持类继承。20 世纪 70 年代，Smalltalk 语言诞生，它取 Simula 的类为核心概念，它的很多内容借鉴于 Lisp 语言。由 Xerox 公司经过对 Smautalk72、76 持续不断的研究和改进之后，于 1980 年推出商品化的，它在系统设计中强调对象概念的统一，引入对象、对象类、方法、实例等概念和术语，采用动态联编和单继承机制。从 80 年代起，人们基于以往已提出的有关信息隐蔽和抽象数据类型等概念，以及由 Modula2、Ada 和 Smalltalk 和等语言所奠定的基础，再加上客观需求的推动，进行了大量的理论研究和实践探索，不同类型的面向对象语言（如：Object-c、Eiffel、C++、Java、Object-Pascal 等）得到了逐步地发展，使得建立了面向对象的方法概念理论体系和研发了实用的软件系统。当前这些面向对象开发语言仍是使用重点，大多数软件的开发设计仍采用面向对象的程序设计方法。

20 世纪 80 年代以来，人们将面向对象的基本概念和运行机制运用到其他领域，获得了一系列相应领域的面向对象的技术。而面向对象方法已被广泛应用于程序设计语言、形式定义、设计方法学、操作系统、分布式系统、人工智能、实时系统、数据库、人机接口、计算机体系结构以及并发工程、综合集成工程等，在许多领域的应用都得到了很大的发展。1986 年在美国举行了首届“面向对象编程、系统、语言和应用（OOPSLA'86）”国际会议，使面向对象受到世人瞩目，其后每年都举行一次，这进一步标志 OO 方法的研究已普及到全世界。

3.2.1.2 面向对象设计基本概述

面向对象的程序设计（Object-Oriented Programming，OOP）实质是把数据和处理这些数

据的过程合并为一个单独的"对象"，而设计思想依赖于一个具有确定特性的、自完备的实体。面向对象程序设计方法具有封装、继承、多态等特征，它属于一种方法论或世界观，要求程序设计者具备一种面向对象的思想基础，即指一种程序设计范型，同时也是一种程序开发的方法。它将对象作为程序的基本单元，将程序和数据封装其中，以提高软件的重用性、灵活性和扩展性。其中：

（1）面向对象的分析方法是利用面向对象的信息来进行构造的建模概念，如实体、关系、属性等，同时运用封装、继承、多态等机制来构造模拟现实系统的方法。而传统的结构化设计方法的基本点是面向过程，即系统被分解成若干个过程。面向对象的方法是采用构造模型的观点，在系统的开发过程中，各个步骤的共同的目标是建造一个问题域的模型。在面向对象的设计中，初始元素是对象，然后将具有共同特征的对象归纳成类，并组织类之间的等级关系，构造类库，在应用时，在类库中选择相应的类。

（2）面向对象设计是使用分析的结果来描述系统如何实现的过程，设计的结果是一组描述系统如何运转的逻辑说明，强调的是一个系统在概念上的软硬件的解决方案，它是 OO 方法中一个环节。其主要作用是对分析模型进行整理，从而生成设计模型提供给 OOP 作为开发依据；一般包括架构设计、用例设计、子系统设计、类设计等，其中架构设计的侧重点在于系统的体系框架的合理性，保证系统架构在系统的各个非功能性需求中保持一种平衡；子系统设计一般是采用纵向切割，关注的是系统的功能划分；类设计是根据通过一组对象、序列图展示系统的逻辑实现。

面向对象程序设计可以被视作一种在程序中包含各种独立而又互相调用的单位和对象的思想，这与传统的思想刚好相反：传统的程序设计主张将程序看作一系列函数的集合，或者直接就是一系列对电脑下达的指令。面向对象程序设计中的每一个对象都应该能够接受数据、处理数据并将数据传达给其他对象，因此它们都可以被看作一个负有责任的角色。

面向对象程序设计推广了程序的灵活性和可维护性，并且在大型项目设计中广为应用。此外，支持者声称面向对象程序设计要比以往的做法更加便于学习，因为它能够让人们更简单地设计并维护程序，使得程序更加便于分析、设计、理解。反对者在某些领域对此予以否认。但这也是正常的，只要支持者和反对者同时存在，才能有效的改善面向对象设计方法和推动面向对象设计的发展。

3.2.1.3　面向对象设计基本概念和属性

一种程序语言离不开基本单元组成，并由这些基本单元实现抽象事物描述，从而能让计算机识别和理解；实质上，软件是问题求解的一种表述形式。而面向对象设计具有独特的封装性、继承性、多态性、抽象性等特征，从而使得它的机能恰好可以表达人们通常的思维方式，并以此来建立问题域的模型，设计出尽可能自然地表现求解方法的软件。

1．对象

对象（Object）是类的实例，是要研究的任何事物。它不仅能表示有形的实体，也能表示无形的（抽象的）规则、计划或事件。对象由数据（描述事物的属性）和作用于数据的操作（体现事物的行为）构成一独立整体。从程序设计者来看，对象是一个程序模块，从用户来看，对象提供所希望的行为，在对内的操作通常称为方法。

2．类

类（Class）定义了一件事物的抽象特点。通常来说，类定义了事物的属性和它可以做到

的（它的行为）。即类是对象的模板，类是对一组有相同数据和相同操作的对象的定义，一个类所包含的方法和数据描述一组对象的共同属性和行为。类是在对象之上的抽象，对象则是类的具体化，是类的实例。类可有其子类，也可有其他类，形成类层次结构。

3．方法

方法（Method）是定义一个类可以做的，但不一定会去做的事。从单纯的方法来看，面向对象设计中的方法有一些类似于面向过程中的函数。并且很多时候，通过方法来实现各对象通信，完成各对象间的消息交互。

4．消息

一个对象通过接受消息、处理消息、传出消息或使用其他类的方法来实现一定功能，这叫做消息传递机制（Message Passing）。即消息一般由三部分组成：接收消息的对象、消息名及实际变元。

5．继承性

继承性（Inheritance）是指在某种情况下，一个类会有“子类”。子类比原本的类（称为父类）要更加具体化，子类会继承父类的属性和行为，并且也可包含它们自己的。这样可大大地减轻在系统实现过程中的重复劳动。同时，在共有属性的基础之上，继承者也可以定义自己独有的特性。当一个类从多个父类继承时，可以称为“多重继承”，多重继承并不总是被支持的，因为它很难理解，又很难被好好使用。

6．封装性

封装（Encapsulation）是一种信息隐蔽技术，它体现于类的说明，是对象的重要特性。封装使数据和加工该数据的方法（函数）封装为一个整体，以实现独立性很强的模块，使得用户只能见到对象的外特性（对象能接受哪些消息，具有哪些处理能力），而对象的内特性（保存内部状态的私有数据和实现加工能力的算法）对用户是隐蔽的。封装的目的在于把对象的设计者和对象者的使用分开，使用者不必知晓行为实现的细节，只须用设计者提供的消息来访问该对象。而具备封装性的面向对象程序设计隐藏了某一方法的具体执行步骤，取而代之的是通过消息传递机制传送消息给它。同时，封装是通过限制只有特定类的实例可以访问这一特定类的成员，而它们通常利用接口实现消息的传入传出。通常来说，成员会依它们的访问权限被分为三种：公有成员、私有成员以及保护成员。有些语言更进一步，如Java可以限制同一包内不同类的访问；C#和VB.NET保留了为类的成员聚集准备的关键字，如internal（C#）和Friend（VB.NET）；Eiffel语言则可以让用户指定哪个类可以访问所有成员。

7．多态性

多态性（Polymorphism）是指由继承而产生的相关的不同的类，其对象对同一消息会做出不同的响应。即对象根据所接收的消息而做出动作，同一消息为不同的对象接受时可产生完全不同的行动，这种现象称为多态性。利用多态性用户可发送一个通用的信息，将所有的实现细节都留给接受消息的对象自行决定，并且同一消息即可调用不同的方法。同时，多态性的实现受到继承性的支持，其利用类继承的层次关系，把具有通用功能的协议存放在类层次中尽可能高的地方，而将实现这一功能的不同方法置于较低层次，这样，在这些低层次上生成的对象就能给通用消息以不同的响应。

8．抽象性

抽象（Abstraction）是简化复杂的现实问题的途径，它可以为具体问题找到最恰当的类

定义，并且可以在最恰当的继承级别解释问题。即是指为了某一分析目的而集中精力研究对象的某一性质，它可以忽略其他与此目的无关的部分。在使用这一概念时，我们承认客观世界的复杂性，也知道事物包括有多个细节，但此时并不打算去完整地考虑它。抽象是科学地研究和处理复杂问题的重要方法。抽象机制被用在数据分析方面，称为数据抽象。数据抽象把一组数据对象以及作用其上的操作组成一个程序实体。使得外部只知道它是如何做和如何表示的。在应用数据抽象原理时，系统分析人员必须确定对象的属性以及处理这些属性的方法，并借助于方法获得属性。

3.2.1.4　面向对象的扩展方法

以上对面向对象设计中所涉及到发展历史和基本概述，以及概念和属性进行了概述[1]。同时，这些内容也是面向对象设计的基本内容，但随着面向对象设计的发展，在 Java、C#等流行语言中也采用诸如委托、接口、命名空间、索引器等概念，这对丰富面向对象设计提供了新的方法，以下以 C#为例来说明委托、接口、命名空间和索引器的编程方法①。

1．委托

委托（Delegate）是一种定义方法签名的类型。当实例化委托时，可以将其实例与任何具有兼容签名的方法相关联，而且可以通过委托实例调用方法。与委托的签名（由返回类型和参数组成）匹配的任何可访问类或结构中的任何方法都可以分配给该委托。方法可以是静态方法，也可以是实例方法。这样就可以通过编程方式来更改方法调用，还可以向现有类中插入新代码。只要知道委托的签名，就可以分配自己的方法。但在方法重载的上下文中，方法的签名不包括返回值。同时，在委托的上下文中，签名的确包括返回值。换句话说，方法和委托必须具有相同的返回值。

委托定义方法为：

```
public delegate int PerformCalculation(int x, int y){};
```

并且委托有如下特点：

（1）类似于 C++ 函数指针，但它们是类型安全的。

（2）允许将方法作为参数进行传递。

（3）可用于定义回调方法。

（4）可以链接在一起；例如，可以对一个事件调用多个方法。

方法不必与委托签名完全匹配。

当委托表示通过其第一个参数关闭的实例方法（最常见的情况）时，委托存储对该方法的入口点的引用和对称是目标对象的引用。当委托表示开放式实例方法时，它存储对该方法入口点的引用。但委托签名必须在其形参表中包括隐藏的 this 参数；在这种情况下，委托不具有对目标对象的引用，必须在调用委托时提供目标对象。

当委托表示静态方法时，委托存储对该方法入口点的引用。当委托表示通过其第一个参数关闭的静态方法时，委托存储对该方法入口点的引用和对目标对象（该对象可分配给方法第一个参数的类型）的引用。当调用该委托时，静态方法的第一个参数接收目标对象。

委托的调用列表就是已排序的委托集，其中列表的每个元素恰好调用该委托所表示的一

① http://msdn.microsoft.com/zh-cn/library

个方法。同时，调用列表可以包含重复的方法。在调用期间，按方法出现在调用列表中的顺序来调用它们。委托尝试调用其调用列表中的每个方法，而重复方法在调用列表中出现一次就调用一次。委托是不可变的；一旦创建，委托的调用列表便无法更改。

委托被称作多路广播委托或可组合委托，因为委托可以调用一种或多种方法，并且可以用在组合操作中。并且合并操作（如 Combine 和 Remove）并不改变现有委托。相反，这样的操作返回一个新委托，其中包含操作结果、未更改的委托或 null。当合并操作的结果是没有引用任何方法的委托时，该操作返回 null。当所请求的操作无效时，合并操作返回未更改的委托。

下面的示例说明如何定义名为 myMethodDelegate 的委托。为嵌套 mySampleClass 类的一个实例方法和一个静态方法创建此委托的实例。实例方法的委托需要 mySampleClass 的一个实例。mySampleClass 实例保存在名为 mySC 的变量中。

```
using System;
public class SamplesDelegate  {
   // Declares a delegate for a method that takes in an int and returns a String.
   public delegate String myMethodDelegate( int myInt );
   // Defines some methods to which the delegate can point.
   public class mySampleClass  {
      // Defines an instance method.
      public String myStringMethod ( int myInt )  {
         if ( myInt > 0 )
            return( "positive" );
         if ( myInt < 0 )
            return( "negative" );
         return ( "zero" );
      }
      // Defines a static method.
      public static String mySignMethod ( int myInt )  {
         if ( myInt > 0 )
            return( "+" );
         if ( myInt < 0 )
            return( "-" );
         return ( "" );
      }
   }
public static void Main()  {
      // Creates one delegate for each method. For the instance method, an
      // instance (mySC) must be supplied. For the static method, use the
      // class name.
      mySampleClass mySC = new mySampleClass();
      myMethodDelegate myD1 = new myMethodDelegate( mySC.myStringMethod );
      myMethodDelegate myD2 = new myMethodDelegate( mySampleClass.mySign
      Method );
      // Invokes the delegates.
```

```
        Console.WriteLine( "{0} is {1}; use the sign \"{2}\".", 5, myD1( 5 ),
        myD2( 5 ) );
        Console.WriteLine( "{0} is {1}; use the sign \"{2}\".", -3, myD1( -3 ),
        myD2( -3 ) );
        Console.WriteLine( "{0} is {1}; use the sign \"{2}\".", 0, myD1( 0 ),
        myD2( 0 ) );
    }
}
```

2. 接口

接口描述的是属于任何类或结构的一组相关功能，且接口和接口成员是抽象的；接口不提供默认实现。接口包含方法、属性、事件、索引器或那些四种成员类型的任意组合。接口不能包含常量、字段、运算符、实例构造函数、析构函数或类型，也不能包含静态成员，而且不能包含任何访问修饰符。当类或结构实现一个接口时，类或结构的所有接口所定义的成员都提供实现。接口本身不提供类或结构，但能够以继承基类功能的方式继承的任何功能。如果是基类实现接口，派生类将继承该实现，这时，派生的类称为隐式实现接口。

接口可以定义为：

```
interface IEquatable<T>
{
    bool Equals(T obj);
}
```

类和结构实现的接口，方式类似于如何类继承基类或结构，有两个例外：

（1）类或结构可以实现多个接口。

（2）当类或结构实现一个接口时，它接收仅在方法名称和签名，因为接口本身包含没有的实现，示例如下：

```
public class Car : IEquatable<Car>
{
    public string Make {get; set;}
    public string Model { get; set; }
    public string Year { get; set; }

    // Implementation of IEquatable<T> interface
    public bool Equals(Car car)
    {
        if (this.Make == car.Make &&
            this.Model == car.Model &&
            this.Year == car.Year)
        {
            return true;
        }
        else
            return false;
```

```
    }
}
```

IEquatable<T> 接口向对象的用户宣布该对象可以确定其是否与同一类型的其他对象相等，而接口的用户不需要知道相关的实现方式。

若要实现接口成员，相应类的成员必须是公共的、非静态和具有相同的名称和与接口成员的签名。属性和索引器的类可以定义额外的属性或索引器在接口上定义的访问器。例如对于接口可能声明一个具有属性的 get 访问器。实现接口的类可以声明一个获得和设置访问器与属性相同。但是，如果属性或索引器使用显式实现，则访问器必须匹配。

接口可以继承其他接口，它有可能继承多的时间，并通过它所继承的基类或通过其他接口继承的接口的接口的类。但是，此类可以实现一个接口只能必须有一个时间，并且仅当该接口声明为在所示的类的定义的一部分（class ClassName : InterfaceName）。如果因为继承一个基类实现该接口的继承接口，它的实现就提供基类；也可能是通过使用虚拟成员实现接口成员的基类。这种情况下派生的类通过重写虚拟成员可以更改接口行为。

这些说明接口有如下特点：

（1）接口是一个抽象基类，如：实现接口的任何非抽象类型必须实现它的所有成员；

（2）不能直接实例化接口；

（3）接口可以包含事件、 索引器、 方法和属性；

（4）接口不包含方法的实现；

（5）类和结构可以实现多个接口；

（6）接口自身可从多个接口继承。

下面示例是显式实现两个接口成员。显式接口实现还允许程序员实现具有相同成员名称的两个接口，并为每个接口成员各提供一个实现。本示例同时以公制单位和英制单位显示框的尺寸。Box 类实现 IEnglishDimensions 和 IMetricDimensions 两个接口，它们表示不同的度量系统。两个接口有相同的成员名称 Length 和 Width。

```
// Declare the English units interface:
interface IEnglishDimensions
{
   float Length();
   float Width();
}
// Declare the metric units interface:
interface IMetricDimensions
{
   float Length();
   float Width();
}
// Declare the Box class that implements the two interfaces:
// IEnglishDimensions and IMetricDimensions:
class Box : IEnglishDimensions, IMetricDimensions
{
```

```
    float lengthInches;
    float widthInches;
    public Box(float length, float width)
    {
        lengthInches = length;
        widthInches = width;
    }
    // Explicitly implement the members of IEnglishDimensions:
    float IEnglishDimensions.Length()
    {
        return lengthInches;
    }
    float IEnglishDimensions.Width()
    {
        return widthInches;
    }
    // Explicitly implement the members of IMetricDimensions:
    float IMetricDimensions.Length()
    {
        return lengthInches * 2.54f;
    }
    float IMetricDimensions.Width()
    {
        return widthInches * 2.54f;
    }
    static void Main()
    {
        // Declare a class instance box1:
        Box box1 = new Box(30.0f, 20.0f);
        // Declare an instance of the English units interface:
        IEnglishDimensions eDimensions = (IEnglishDimensions)box1;
        // Declare an instance of the metric units interface:
        IMetricDimensions mDimensions = (IMetricDimensions)box1;
        // Print dimensions in English units:
        System.Console.WriteLine("Length(in): {0}", eDimensions.Length());
        System.Console.WriteLine("Width (in): {0}", eDimensions.Width());
        // Print dimensions in metric units:
        System.Console.WriteLine("Length(cm): {0}", mDimensions.Length());
        System.Console.WriteLine("Width (cm): {0}", mDimensions.Width());
    }
}
```

3. 索引器

索引器允许类或结构的实例就像数组一样进行索引，类似于属性，不同之处在于它们的访问器所采用的参数。通常索引器具有以下特征：

（1）使用索引器可以用类似于数组的方式为对象建立索引；

（2）get 访问器返回值，set 访问器分配值；

（3）this 关键字用于定义索引器；

（4）value 关键字用于定义由 set 索引器分配的值；

（5）不必根据整数值进行索引，并决定如何定义特定的查找机制；

（6）可被重载；

（7）可以有多个形参，例如当访问二维数组时；

（8）在语法上方便创建客户端应用程序，可将其作为数组访问的类、结构或接口，并经常是用于封装内部集合或数组类型中实现的。同时，索引器类型及其参数类型必须至少如同索引器本身一样是可访问的。索引器的签名由其形参的数量和类型组成，它不包括索引器类型或形参名，如果在同一类中声明一个以上的索引器，则它们必须具有不同的签名；索引器值不属于变量，因此，不能将索引器值作为 ref 或 out 参数进行传递；要为索引器提供一个其他语言可以使用的名字，需要使用声明中的 name 特性。

下面的示例说明如何声明私有数组字段、temps 和索引器。使用索引器可直接访问实例 tempRecord[i]。另一种使用索引器的方法是将数组声明为 public 成员并直接访问它的成员 tempRecord.temps[i]。

```
class TempRecord
{
    // Array of temperature values
    private float[] temps = new float[10] { 56.2F, 56.7F, 56.5F, 56.9F, 58.8F,
                                        61.3F, 65.9F, 62.1F, 59.2F, 57.5F };
    // To enable client code to validate input
    // when accessing your indexer.
    public int Length
    {
        get { return temps.Length; }
    }
    // Indexer declaration.
    // If index is out of range, the temps array will throw the exception.
    public float this[int index]
    {
        get
        {
            return temps[index];
        }
        set
        {
            temps[index] = value;
        }
    }
}
class MainClass
```

```
{
    static void Main()
    {
        TempRecord tempRecord = new TempRecord();
        // Use the indexer's set accessor
        tempRecord[3] = 58.3F;
        tempRecord[5] = 60.1F;
        // Use the indexer's get accessor
        for (int i = 0; i < 10; i++)
        {
            System.Console.WriteLine("Element #{0} = {1}", i, tempRecord[i]);
        }
        // Keep the console window open in debug mode.
        System.Console.WriteLine("Press any key to exit.");
        System.Console.ReadKey();
    }
}
```

在使用索引器时，有时需要提高索引器的安全性和可靠性，通常采用如下方法来实现：

（1）确保结合某一类型的错误处理策略，以处理万一客户端代码传入无效索引值的情况，也可以将错误处理代码放入索引器自身内部。以确保为用户记录在索引器的访问器中引发的任何异常。

（2）应当为 get 和 set 访问器的可访问性设置尽可能多的限制，这一点对 set 访问器尤为重要。

4. 命名空间

命名空间（Namespace）表示标识符（identifier）的上下文（context）。一个标识符可在多个命名空间中定义，它在不同命名空间中的含义是互不相干的。这样，在一个新的命名空间中可定义任何标识符，它们不会与任何已有的标识符发生冲突，因为已有的定义都处于其他命名空间中。这一特点是使用命名空间的主要理由，在大型的计算机程序或文档中，往往会出现数百或数千个标识符。命名空间（或类似的方法）提供一隐藏区域标识符的机制。通过将逻辑上相关的标识符组织成相应的命名空间，可使整个系统更加模块化，最终以命名空间为基础的标识树来进行统一管理。

而命名空间在编程语言中，它是一种特殊的作用域，包含了处于该作用域内的标识符，且本身也用一个标识符来表示，这样使将一系列在逻辑上相关的标识符用一个标识符组织了起来。许多现代编程语言都支持命名空间。在一些编程语言（例如 C++和 Python）中，命名空间本身的标识符也属于一个外层的命名空间，也即命名空间可以嵌套，构成一个命名空间树，树根则是无名的全局名空间。

在 C#中，namespace 关键字用于声明一个范围。在项目中创建范围的能力有助于组织代码，并可以创建全局唯一的类型。在下面的示例中，名为 SampleClass 的类在两个命名空间中定义，其中一个命名空间嵌套在另一个之内。

```
namespace SampleNamespace
{
```

```
    class SampleClass
    {
        public void SampleMethod()
        {
            System.Console.WriteLine(
              "SampleMethod inside SampleNamespace");
        }
    }
    // Create a nested namespace, and define another class.
    namespace NestedNamespace
    {
        class SampleClass
        {
            public void SampleMethod()
            {
                System.Console.WriteLine(
                  "SampleMethod inside NestedNamespace");
            }
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            // Displays "SampleMethod inside SampleNamespace."
            SampleClass outer = new SampleClass();
            outer.SampleMethod();

            // Displays "SampleMethod inside SampleNamespace."
            SampleNamespace.SampleClass outer2 = new SampleNamespace.Sample
            Class();
            outer2.SampleMethod();
            // Displays "SampleMethod inside NestedNamespace."
            NestedNamespace.SampleClass inner = new NestedNamespace.Sample
            Class();
            inner.SampleMethod();
        }
    }
}
```

3.2.2 面向构件设计方法

构件是一个可以独立发布的功能部分，通过接口访问服务，具有相对独立的功能和复用价值。构件按其复用粒度的大小和关注点分为业务构件和服务构件，图 3-1 所示是对构件定

义的表述，从图中容易得出，构件有三个重要的方面：

（1）它将构件定义为一个可交付的软件单元；

（2）构件会提供一些有用的功能，这些功能集合到一起能够满足一些软件需求，且功能的设计符合一些设计准则；

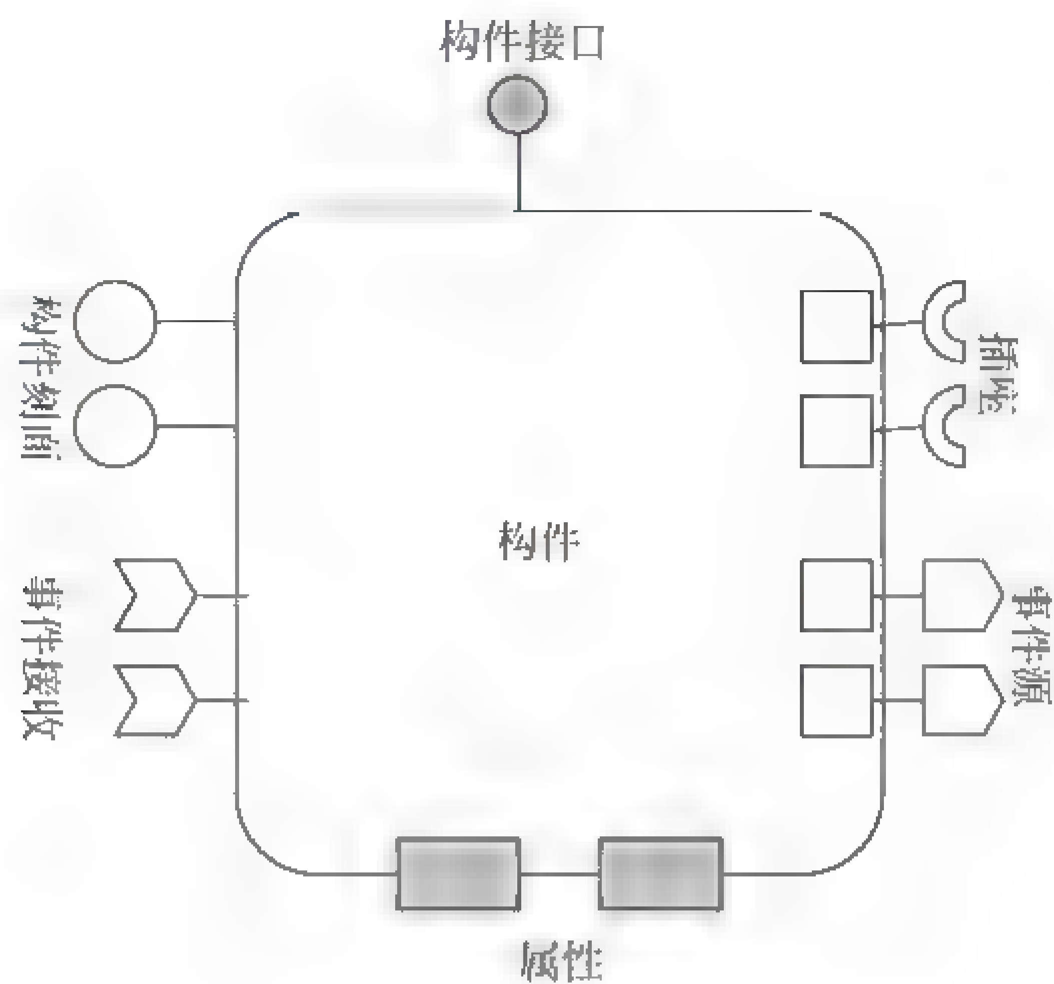

图 3-1　构件定义

（3）构件通过接口提供服务。

而基于构件的软件开发（Component-Based Software Development，CBSD）是一种基于分布对象技术、强调通过可复用构件设计与构造软件系统的软件复用途径。即是一种基于预先开发好的软件构件，通过将其集成组装的方式来开发新的软件系统的方法；又称基于构件的软件工程（CBSE），它是软件复用的实现方式之一，其根本目的仍然是为了提高软件开发的质量和效率。

在实现基于构件的软件开发时，构件必须有一个关于它所提供的服务的抽象描述，以作为服务的客户方和提供方之间的契约，这就是构件接口。不同构件（包括不同厂商的构件）之间可以相互协作，这种协作是通过构件接口进行的，如图 3-2 所示。

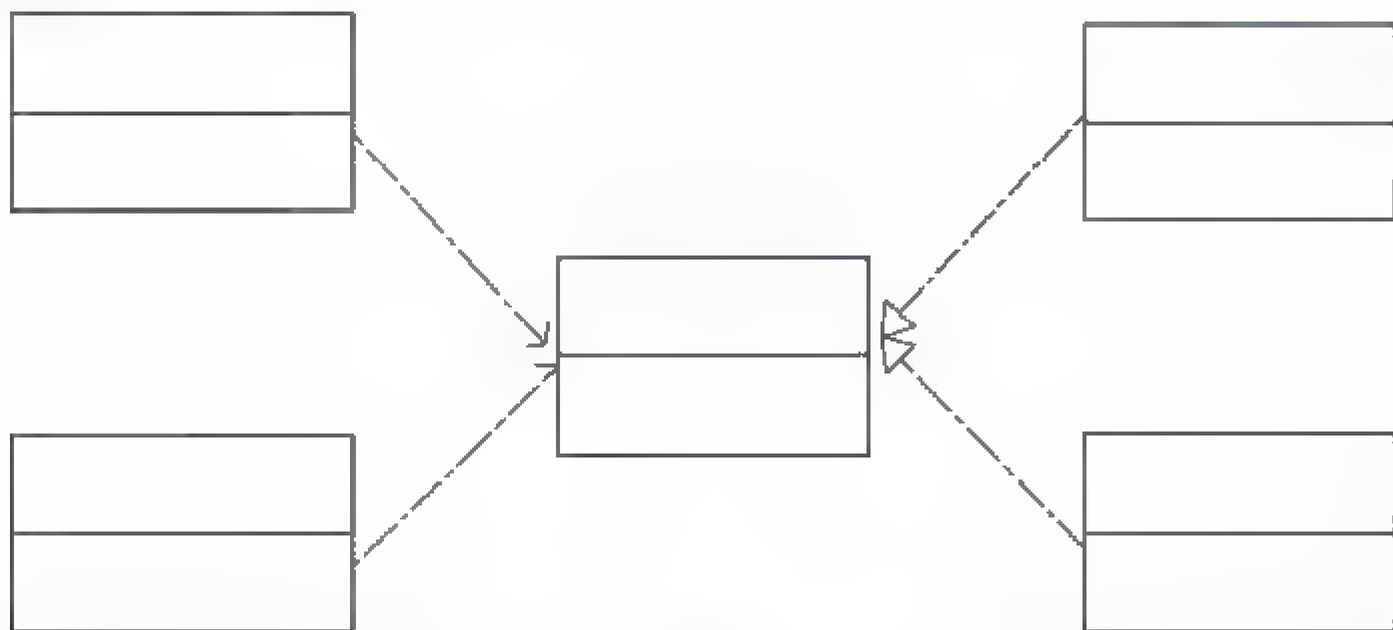

图 3-2　构件接口定义

从图中可知，左边部分表示不同的构件使用者，图中右边部分表示不同开发厂商创造的构件。使用者和厂商之间的连接点是构件接口，对于构件厂商而言，他们只关注于对构件接

口的功能实现；而对构件的使用者来说，则仅关注于其依赖的构件接口。同时将构件接口和构件分开，这是面向构件方法论的基本要求，也是不同构件（包括不同厂商的构件）可装配、可替换、可组合的基础，构件接口也是构件复用的基础。

3.2.2.1 基于构件的软件开发应用背景

基于构件的软件开发方法（CBSD）的兴起主要是源于如下不同的背景[①]：

（1）在学术研究方面对现代软件工程思想，特别是对软件复用技术的高度重视；

（2）在技术研发方面所取得的有效进展，虽然缺少理论的支持，但在图形用户界面（GUI）和数据库应用中基于部件的组装技术的成功应用；

（3）一些主流互操作技术开发者的积极推动，如 OMG 的 CORBA / CCM、微软公司的 COM/DCOM 以及 SUN 公司的 EJB 已成为主流的构件实现规范，相应的软件中间件平台规范也已获得较为普遍的接受；

（4）由于面向对象技术的广泛使用，提供了构作和使用构件的概念基础和实用工具，事实上，主流的构件实现模型均基于对象技术；

（5）需求的变化促使需要有一种可得用性好、便于操作的软件开发方法来促使新的软件研发方法出现，目前最著名的上海普元构件、北大青鸟构件等。

从开发方法的角度来看，CBSD 提供了一种自底向上的、基于预先定制包装好的类属元素（构件）来构作应用系统的途径。CBSD 的发展和中间件技术的发展是密切相关的，正是中间件技术及其平台提供了构件开发和构件组装的技术基础和机制。因此，当前 CBSD 讨论的重点主要局限于基于 COM/DCOM、CORBA/CCM 和 EJB 等主流规范的二进制级构件。从复用的角度看，CBSD 支持的是黑盒、组装式复用方式。系统开发者不能对构件进行源代码级的修改，最多只能是通过参数方式进行适应性调整。构件的组装在中间件平台上进行，构件间通过中间件提供的通信协议和设施进行交互。

CBSD 的主要活动包括：构件获取、构件认证、构件适应性修改、构件组装和应用部署。构件获取根据应用需求，去寻找符合或基本符合需求的构件，通常途径有以下三种：

（1）来自第三方构件开发商；

（2）货架商品式的（COTS）构件；

（3）集成者根据特殊应用需求开发。

构件认证对候选构件进行可用性、可信性和复用历史等方面的考察，以确定是否可用于新应用系统中。认证可由集成者自己完成，也可委托第三方认证。

构件适应性修改由于应用的特殊性，有时需要对候选构件进行适当修改。这是因为 CBSD 所支持的黑盒复用模式，集成者只能通过参数调整的方式修改构件。如要涉及代码级的修改，通常仍需由原构件开发商完成。

构件组装根据应用的功能及非功能需求，将相关的构件组装在一起，主要工作涉及一些胶合代码和构件间连接关系描述信息的生成。

应用部署将构成应用系统的各构件或构件组分别部署到相应的运行环境中，也即是将组装形成的应用系统“安装”到相应中间件平台上。当部署完毕，应用系统方可投入运行。

---

① http://iknow.seforge.org/sewiki/index

基于 CBSD 正在受到越来越多的关注，并在很多领域实施成功应用。然而，CBSD 的有效性和高效性还面临一些挑战。CBSD 当前主要着重于目标码层次的互操作能力，缺少涉及需求和设计层次的系统化方法学支持，因此，需要和软件复用的已有研究成果进行无缝整合。同时，大量可供使用的构件的存在是 CBSD 能够被广泛应用的前提，而这正是目前大多数应用领域所欠缺的。此外，多种构件实现规范及体系结构风格的共存，以及通过需求自动识别构件进行组装成新的软件系统，使得软件体系结构失配成为 CBSD 方法的另一个难题。

3.2.2.2　基于构件的软件开发的属性

构件作为一个相对较独立的软件实体，怎样通过构件实现软件开发？这时就需要对构件建立一种人们都能接受的属性来统一规范、应用这些构件。图 3-3 所示是基于构件的开发过程，图 3-4 所示是构件规范特征。

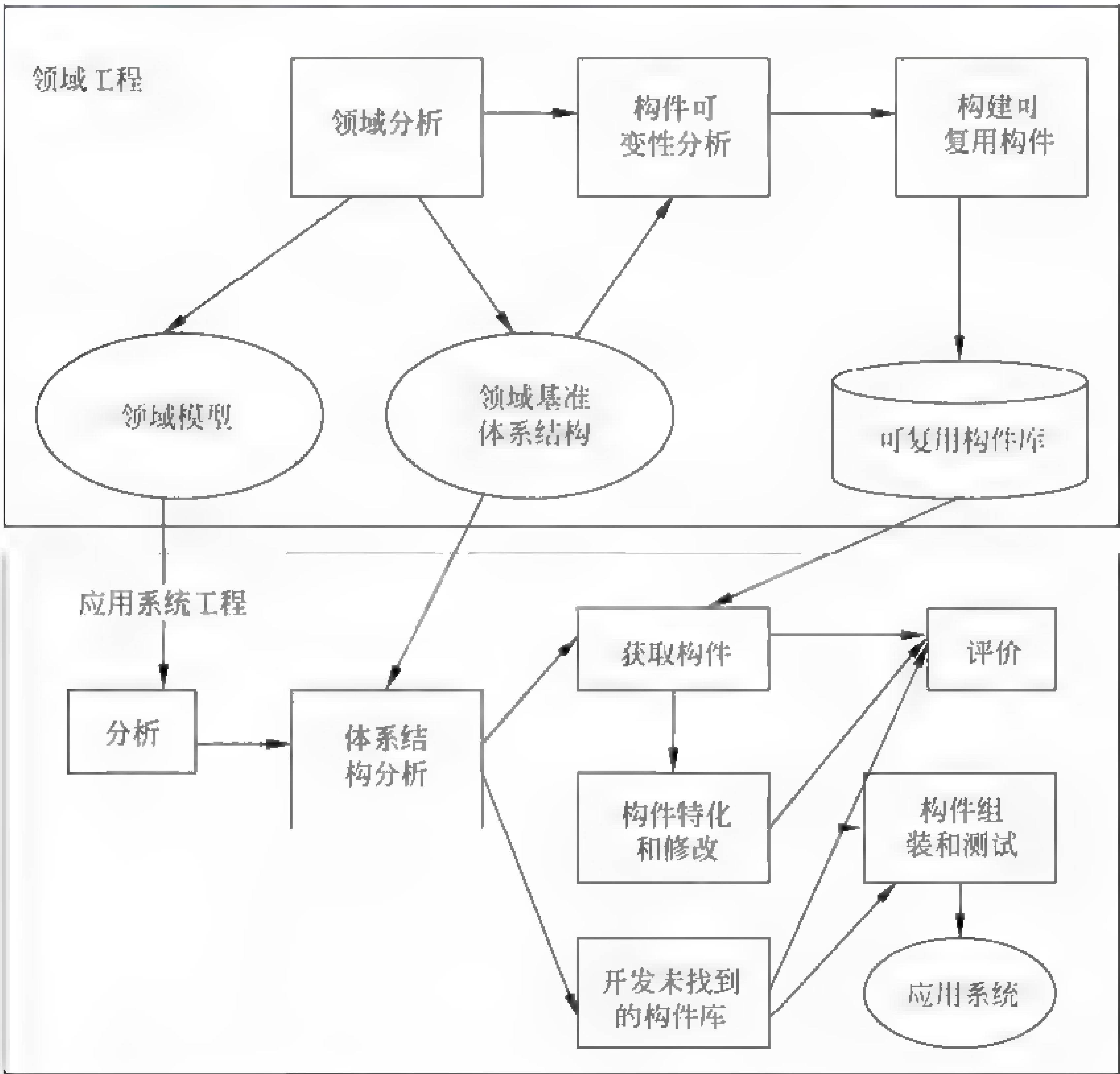

图 3-3　基于构件的软件开发过程

在图 3-3 中，领域工程表示不同的领域有不同的构件来支撑，这是由于不同领域的需求有表达的构件功能是不一样的，所反映出的构件属性也是不一样的。

1．规格说明

建立在构件接口概念之上，作为服务提供方与客户方之间的契约，以便为构件操作带来可识别的规范。

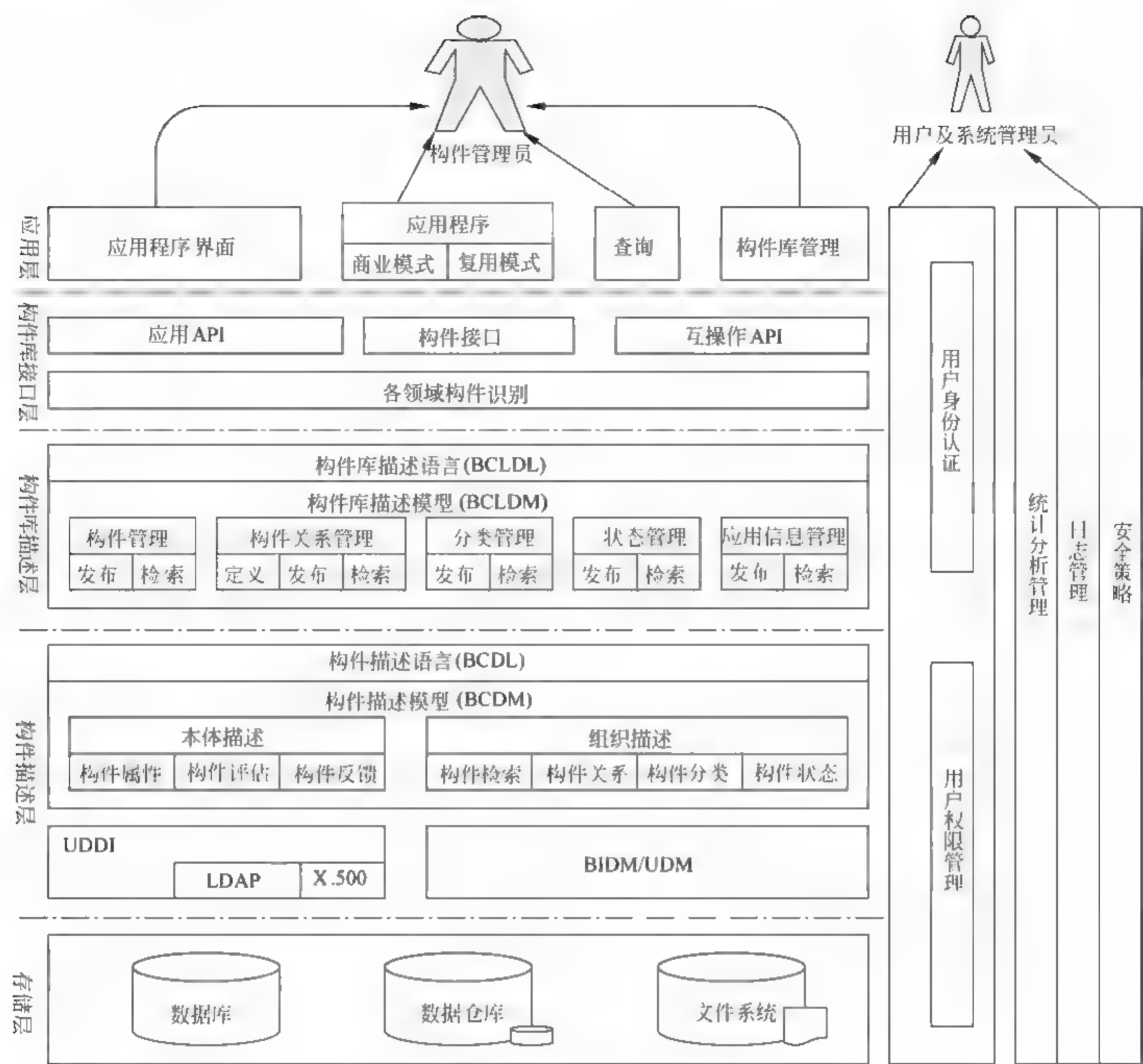

图3-4　构件规范层次描述图

2．构件的概念、内容和上下文

（1）概念（concept）：关于“构件做什么”的抽象描述，可以通过概念去理解构件的功能。概念包括接口规约和语义描述两部分，语义描述和每个操作相关联（至少表示为前后置谓词形式）。

（2）内容（content）：概念的具体实现，描述构件如何完成概念所刻画的功能。

（3）上下文（context）：描述构件和外围环境在概念级和内容级的关系，刻画构件的应用环境，为构件的选用和适应性修改提供指导。

3．构件刻面

对领域进行分析，所得到的一组基本的描述特征，且可以描述构件执行的功能、所操作的数据、构件应用的周境或任何其他特征等。通常构件刻面由抽象（它是构件概念的抽象性描述）、操作（它是构件所提供的操作的描述）、操作对象（它描述操作的对象）、依赖（它描述构件与外界的依赖关系）等部分构成。

4．构件领域性

需要进行领域分析，收集、分析不同需求领域中的公共部分和相似部分，以及不同的部分，从而建立基准的构件结构模型，并在这个模型上实现裁剪和扩展，从而到达领域构件的可复用性。

5．构件的泛化和可变分析

提高其通用性，同时寻找候选构件在不同应用中的变化点（Variation Point），通过设置参数、继承或其他手段，使可变部分局部化。

6．构件可复用

构件的可复用是构件的突出的优点，并利用构件的可复用性按需构造新的软件系统。

7．构件表现形式

构件库和构件库管理系统。其中，构件库是指持久存储的可复用软件构件及其相关资源的集合；构件库管理系统是指对构件库进行统一管理和控制的软件系统，且对软件构件进行管理，并对其复用提供支持的基础设施。

8．构件可定制性

构件的定制性就是按需求进行构件可复用，并能构造出供用户使用的软件系统。

9．构件的生命过程

根据标准 IEEE 1517 将构件生命周期的过程、活动和任务组织包含基本过程类、跨项目过程类、组织过程类和支持过程类等四部分。

10．构件的资产管理

包括构件的本身构件库管理、配置管理等。它的目的是解决软件复用在管理、保存、检索、版本控制、变化控制和功能发布中的特殊需求；保证资产的潜在使用者能够知道资产的存在，能够容易地找到最新版本的资产，能够容易地理解资产的用处、状态和质量。

3.2.2.3　基于构件的软件开发的核心特征

基于构件的软件开发方法是一种软件研发技术，是以层次粒度和关注点分离为基础，根据用户需求，通过构件接口、适配器等完成构件组装。构件组装实现了更低层次的代码和逻辑实现，而面向构件的软件开发方法则在此基础上开创了全新的应用软件的开发模式，并重点关注和解决现代应用软件开发中的三个核心关键问题：全流程管理（Process）、无缝访问各层资源（Access）、易于根据需求而改变（Change）。

构件的应用流程是通过构件组装来完成的，而且是支持全流程的实现。全流程也就是对于三层不同逻辑资源的一体化使用：

1．代码逻辑（Code Logic）

又称为功能逻辑，可以支持用各种语言（Java，C++，BPEL…）编写的代码逻辑的流程，它们一旦被封装到标准的服务构件中，就可以被用来组装成更高级的业务构件。而在业务组装开发环境中，原来的实现技术和语言已经被屏蔽了，这时对于应用开发者来讲具体的实现技术和语言已经变得透明和不再重要了。

2．构件逻辑（Component Logic）

又称应用逻辑，可以支持到基于服务构件的逻辑组装，从而开发出更粗粒度的业务构件，

让业务构件实现了具体的业务模块和流程。

3．服务逻辑（Service Logic）

又称集成逻辑，可以在服务构件或是业务构件中访问和使用外界的服务资源，以实现更高层次的业务构件。

对于各种资源的无缝访问是现代应用的需要，而构件正是实现了标准的无缝访问。无论是访问逻辑/流程，还是访问数据/信息，甚至是访问页面/展现。构件可以封装任何已有的和新建的模块功能，通过暴露构件的服务可以被更多的使用者访问。构件提供的标准化访问能力使在具体应用中遇到的集成问题得到迎刃而解，使得过去那些专有化的集成和访问方式将一去不复返了，取而代之的是构件和服务的与生俱来的无缝访问和集成能力。

3.2.2.4 基于构件的软件开发的实施策略

面向构件的软件开发的过程中，首先需要构件库和构件库管理系统存在，且针对不同的需求，存在不同领域的构件，比如电信行业构件，证券行业构件等，即不同的需求领域需要不同的领域构件库和领域构件管理系统。

1．明确基于构件的软件开发步骤

通常在已存在的构件库和构件库管理系统中，按分析用户需求、查询构件、获取构件、构件组装和完成应用系统部署等流程来明确构件的软件开发步骤，而在这个过程包括构件发布、构件检索、构件分类、构件存储和构件配置等操作。

2．确定可复用性构件

基于构件的软件开发主要特征之一就是提高构件的复用性，提高软件开发效率，使较大的可复用构件带来更多的回报和效益。

3．构件选择策略

构件的选择对提高构件的复用至关重要，这是因为构件是软件，就需要考虑构件的健壮性、稳定性、可用性等。下面是构件选择步骤：

（1）评估构件可能被复用的次数，优先选择可能被多次复用的构件；

（2）评估可复用构件在系统项目中的重要性；

（3）估计建立/获取/准备可复用构件的成本：若构件已存在，则估计将它改变为可复用构件所需的时间、精力和成本；

（4）估计构件的可能被复用的使用期限：优先选择建立时间短、使用期限长的可复用构件；

（5）估计维护和管理构件的成本：估计构件必须被复用多少次，才能回收投资。

4．参考构件的文档描述

通过参考构件的文档描述，可以方便获得构件可复用性的状态和特征，通常这些文档包括：名称描述（解释和基于目录说明的简短描述）、分类描述（根据分类模式，对构件所做进行分类的说明）、领域描述（适合于该构件应用的领域的描述）、使用信息（应包含性能限制、法律限制等限定信息）、测试信息（包括测试计划、测试数据和预期的测试结果等）、质量说明（例如出错率、文档的质量和遵循标准的程度）、演化说明（提升构件的复用层次所推荐的改进之处）和历史信息（有关构件的历史复用信息）等。

3.2.3　UML 建模方法

统一建模语言（Unified Modeling Language，UML）是非专利的第三代建模和规约语言，是用来对软件密集系统进行可视化建模的一种语言。UML 是一种开放的方法，用于说明、可视化、构建和编写一个正在开发的、面向对象的、软件密集系统的制品的开放方法。UML 展现了一系列最佳工程实践，这些最佳实践在对大规模、复杂系统进行建模方面，特别是在软件架构层次已经被验证有效的系统中。UML 被 OMG（Object Management Group）采纳作为业界的标准，UML 最适于数据建模，业务建模，对象建模，组件建模。它集成了 Booch，OMT 和面向对象软件工程的概念，将这些方法融合为单一的，通用的，并且可以广泛使用的建模语言。UML 打算成为可以对并发和分布式系统的标准建模语言。但 UML 并不是一个工业标准，但在 OMG 的主持和资助下，UML 正在逐渐成为工业标准。OMG 之前曾经呼吁业界向其提供有关面向对象的理论及实现的方法，以便制作一个严谨的软件建模语言（Software Modeling Language）。目前有很多业界的领袖亦真诚地回应 OMG，帮助建立一个业界标准。

3.2.3.1　UML 背景及概况

在 1990 年时，软件设计方法的早期版本已经对对象模式和相关技术有着浓厚的兴趣，基于这个模式的新的编程语言（比如 Smalltalk, Eiffel, C++, 和 Java）已经被设计并投入使用，伴随着这些语言出现的还有难以理解的面向对象[Object-Oriented（OO）]软件设计方法和建模符号。并考虑到对象模式包含了基本概念的相对较小的子集（包括封装、继承和多态），在这些方法中存在非常多的重叠和概念上的结合，即大多数情况下是由于符号性的和其他并不重要的差异而使其变得很模糊不好理解。这样就导致了令人难以理解以及不必要的市场分歧，反过来也阻碍了有实用价值的新模式的采用。这时软件开发者很难在这些相互矛盾的语言，工具，方法和供应商中做出选择。基于此，Rational 软件提出了统一建模语言（UML）的初始版，并在 Grady Booch、Ivar Jacobson 和 Jim Rumbaugh 的领导下得到了快速和积极的反响，他们的目的并不是为了提出任何新的内容，而是过高级领域思想领导者们的协作把各种各样的 OO 方法的最好特性添加到一个单独的和与供应商无关的模型化语言和注释中。这样 UML 很快地成为了一个广泛的实践标准。随着 1996 年对 OMG 对它的采用，UML 成为了一个广泛被接受的行业标准，OMG 并在 1997 年发布了统一建模语言 UML，它的目标之一就是为开发团队提供标准通用的设计语言来开发和构建计算机应用。但到了 2003 年，UML 才真正获得了业界的认同。

1994 年 10 月，Grady Booch 和 Jim Rumbaugh 开始致力于这一工作。他们首先将 Booch 93 和 OMT-2 统一起来，并于 1995 年 10 月发布了第一个公开版本，称为统一方法 UM 0.8（Unitied Method）。

1995 年秋，OOSE 的创始人 Ivar Jacobson 加盟到这一工作。经过 Booch、Rumbaugh 和 Jacobson 三人的共同努力，于 1996 年 6 月和 10 月分别发布了两个新的版本，即 UML 0.9 和 UML 0.91，并将 UM 重新命名为 UML（Unified Modeling Language）。

1996 年，相关机构和组织认这 UML 作为其商业策略已日趋明显。UML 的开发者得到了来自公众的正面反应，并倡议成立了 UML 成员协会，以完善、加强和促进 UML 的定义工作。当时的成员有 DEC、HP、I-Logix、Itellicorp、IBM、ICON Computing、MCI Systemhouse、

Microsoft、Oracle、Rational Software、TI 以及 Unisys。这一机构对 UML 1.0（1997 年 1 月）及 UML 1.1（1997 年 11 月 17 日）的定义和发布起了重要的促进作用。UML 是一种定义良好、易于表达、功能强大且普遍适用的建模语言，并且作用域不限于支持面向对象的分析与设计，还支持从需求分析开始的软件开发的全过程。

2003 年 3 月，由 OMG 发布了 UML 1.5；接着 2004 年 4 月，OMG 又发布了 UML 2.0，它在以下五方面对以前的版本进行了改进[①]：

1．在语言定义方面精确程度有了相当的提高

这就是支持自动化高标准需要的结果，此标准是 MDD（Model-Driven Developement）所必须的，自动化意味着模型（以及后来的模型语言）的不明确和不精密的消除，所以计算机程序能转换并熟练地操纵模型。

精确程度上主要是引入了元模型、语义来描述 UML，使整个 UML 的精度增强，即：

（1）用元模型来架构 UML。UML 2.0 架构是由一组低层次的建模概念和模式所组成，它们在大多数的案例中要么过于初级，要么太抽象，以至于不能直接地在建模软件应用程序中使用。然而，它们相对的简单性使得它们在其语义和形成的规则上更加精确。这些语义与形式化概念，以不同的方法结合产生了更加复杂的用户级别的建模概念。例如，在 UML 1 中，在所有权角度上（元素包含另外一些元素），命名空间的概念（又称唯一命名的元素集合）与分类器的概念（元素是能根据它们的属性进行分类的）上，都与单个的复杂语义概念绑定在一起。而在 UML 2.0 架构中，这些概念被分离开，并且它们的语法和语义也被单独的定义。

（2）可扩展和更加精确的语义描述。UML 1.0 模型概念语义的定义在许多方法都是存在的问题，它所描述的层次有些地方具有某些广泛的和详细的描述（例如，状态机），但是非常不平均，而其他的一些地方几乎没有解释。UML 2.0 规范主要强调了语义，尤其是在基本行为动态的关键领域中。

（3）一种清晰定义的动态语义框架。UML 2.0 规范澄清了一些在老版本中的严重语义缺陷。图 3-5 所示描述了这个框架，主要内容包括：在运行期间的链接和实例的结构化语义、结构和行为之间的关联，以及语义的基础或因果关系模型通过所有当前在 UML 中的高级行为形式（即状态机，活动，交互）所共享。这同样也确保了那些通过不同的形式表达行为的对象可以相互的交互。

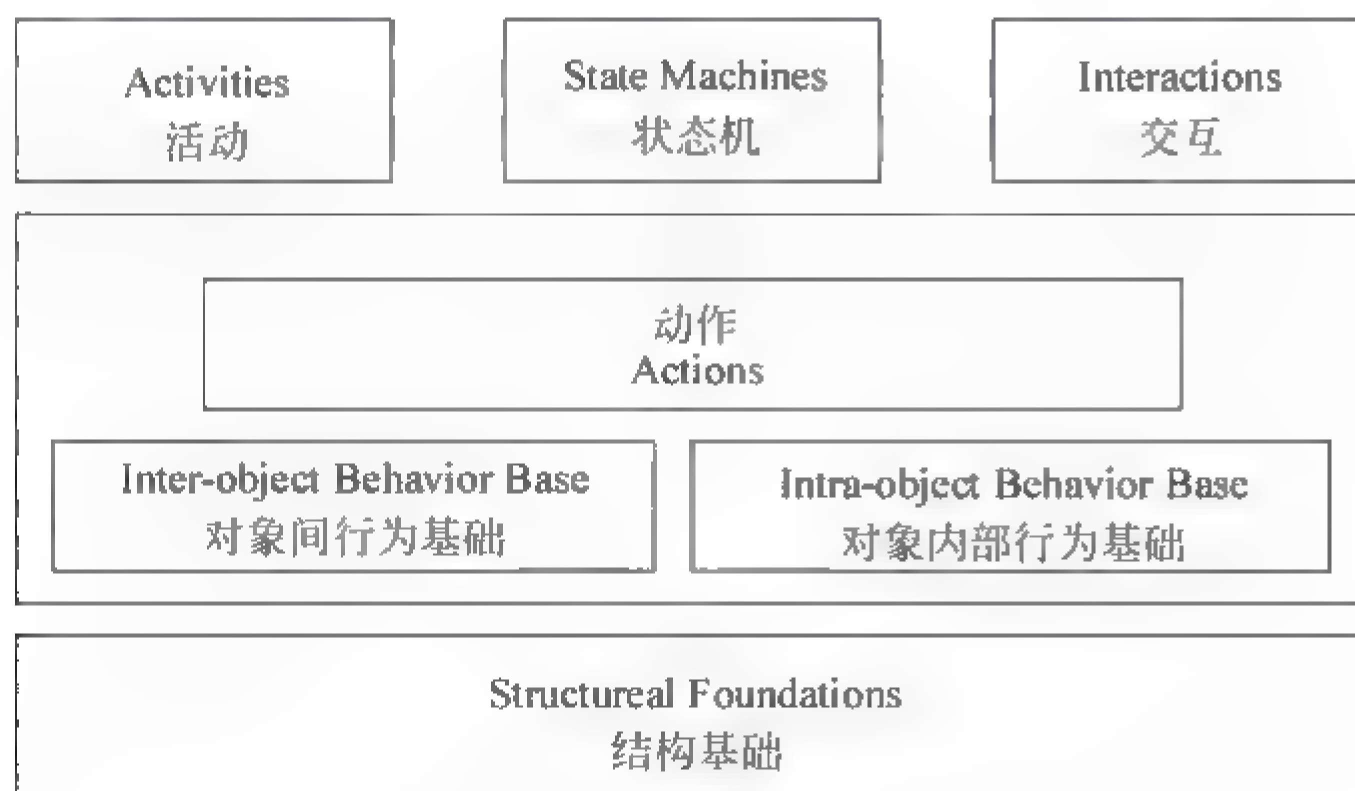

图 3-5 UML 2.0 语义框架

① http://www.ibm.com/developerworks/cn

2. 一种改良的语言组织

其特性是由模块化决定的，模块化的特点在于它不仅使得语言更加容易的被新用户所采用，而且促进了工具之间的相互作用。

UML 2.0 在某种程度上进行了模块化，允许有选择性的使用一些语言模块，以便解决语言复杂度的问题。表 3-1 所示就是 UML 2.0 的语言单元。

表 3-1　UML 2.0 的语言单元

序号	语言单元	目的
1	动作	（基础）细粒度动作的建模
2	活动	数据和控制流行为建模
3	类	（基础）基本结构的建模
4	组件	组件技术的复杂结构建模
5	部署	部署建模
6	通用行为	（基础）公共行为语义基础和时间建模
7	信息流	抽象数据流建模
8	交互	内部对象行为建模
9	建模	模型组织
10	Profiles	语言定制化
11	状态机	事件驱动行为建模
12	结构	复杂的结构建模
13	模板	模式建模
14	用例	非正式的行为需求建模

3. 重点改进大规模的软件系统模型性能

一些流行的应用软件表现出将现有的独立应用程序集成到更加复杂的系统中去。这是一种趋势，它将可能会继续导致更加复杂的系统。为了支持这种趋势，将更加灵活和新的分等级的性能添加到语言中去，用以支持软件模型在任意复杂的级别中使用。

实际上，新的建模能力的实质是对已存在的特征进行简单地扩充，以使用于大规模的软件系统的建模。此外，这些扩展都是通过使用相同的基本方法达到的，即在不同的抽象层面上递归地应用那些相同的基本概念集。即可以把一个给定类型的模型元素合并到单元里，依次类推，可以用这种方式去实现在抽象层面上的合并，并把这些合并后的单元作为一个模块进行使用。这跟编程语言中的过程类似，它能根据想要的深度进行嵌套的调用。主要从复杂结构、活动、交互和状态机四个角度来提高大规模软件系统的模型性能。

（1）复杂结构。特征的依据来自于对不同架构描述语言长期使用的经验。这些语言的特点在于它们通过相对简单图的概念进行描述：基本的结构性结点，也就是所谓的部件，它们可以有一个或多个端口，它们之间可以由又称连接器的通信通道进行连接。同时，这些内容可以被封装成更高层的单元，依次类推，这些新封装成的单元也可以有自己的端口以便于与其他更高层的单元合并成更高层的单元。这些概念在 UML 1.0 中对于协作的定义里可以找到，但不能递归。

（2）活动。在 UML 中，活动被用来对不同种类的流程建模，包括有信号流或数据流，也有算法流或过程流。但在 UML 1.0 中，行为建模在流的类型方面有大量严格的限制，并且

这其中的很多限制都是由于在基本的状态机的顶部行为被覆盖了，所以，它们受限于状态机的语义。在 UML 2.0 中提出了异议，因此在 UML 2.0 中用一个消除了这些限制的更泛化的语义基础替代了状态机的底层。此外，这些语义基础也从很多行业标准和业务过程形式中得到灵感，其中包括 BPEL4WS（Business Process Executable Language for Web Service）在基础的形式上增加了一系列非常丰富并且非常精确的建模特征。同时，活动也包括中断的活动流、复杂形式的并发控制和多样的缓冲配置等能力。

（3）交互。在 UML 1.0 中，交互性是由协作图中序列消息的注释或单独的序列图来表现的。但失去了对序列进行重用的能力和对不同的复杂控制流充分建模的能力。而在 UML 2.0 中，把交互性作为单独命名的建模单元进行引入，这样的交互性表现在内部对象间任意复杂的通信，而且甚至可以被参数化以用来描述上下文独立的交互模式。

（4）状态机。基本思想是它可以创建一个复合状态的完整模块，它具有清晰的转换入口点和出口点。反过来，研发人员也可以通过一个离散的和可重复使用的状态机的规范来分别地定义上述的复合状态的内在分解。也就是说，在这个状态机或某些其他的状态机中，相同的规范可以在多处重复使用,这样就使得在不同上下文中的共享行为模式的规范更加简单化。

4．对特定领域的改进的支持

使用 UML 的实践经验证明了其所谓的“扩展”机制的价值，这些机制被统一化，精炼化后，使得基础语言更加简化，更加准确精炼。

UML 1.0 的实践应用表明 UML 的一个相当通用的方式是首先为一个特定的问题或领域定义一个 UML Profile，然后用这个 Profile 代替普通的 UML。实质上，这些 Profile 就是一种生成像特定领域语言（DSL）的方法，并且使用 UML Profile 的一种可选择的办法是使用 MOF（Meta Object Facility）标准和工具定义一种新的自定义模型化语言。但是一个 UML Profile 必须与标准 UML 保持一致；换句话说，一个 UML Profile 限定了标准 UML 的概念，这种限定是通过定义上的限制来限制给它们提供一种唯一的特殊领域解释的概念。

因此，在 UML 2.0 中的 profiling 机制已经被合理化且它的性能也已经被扩展了；在原型和 UML 概念之间的连接已经被扩展了。事实上，好像一个 UML 2.0 原型被定义成只是一个现有 UML 元类的子集，并带有关联的属性（代表加有标签的值的标签）、操作和限制。并且 UML 2.0 profiling 机制同时也可以用作一种机制，它可以从多种不同的域中观察到一个复杂的 UML 模型。也就是说，任何一个 profile 都可以有选择性的以任何方式被应用或是不被应用，只要不影响基础的 UML 模型。

5．全面的合并，合理化、清晰化各种不同的模型概念

从而使得一种单一化，更加统一化语言的产生。

在 UML 1.5 中介绍过动作，动作的概念上的模型被特意的普通化，从而提供数据流和控制流计算模型。这就导致了与活动模型在概念上非常相似。UML 2.0 利用了这种相似，它为动作和活动提供了一个通用的在语法上和语义上的基础。从研发人员角度来看，在不同层次上的抽象显得有些过于的形式主义，因为它很典型模拟了不同层次之间存在的现象。

6．一款 UML 开源软件——ArgoUML

ArgoUML 是一种用于系统设计和架构设计的、优秀的开放源代码工具，当前版本是 032.2，运行图如图 3-6 所示。它用 Java 构造，并遵守开源软件的 BSD 协议。能运行在任何支持 Java 的平台上。

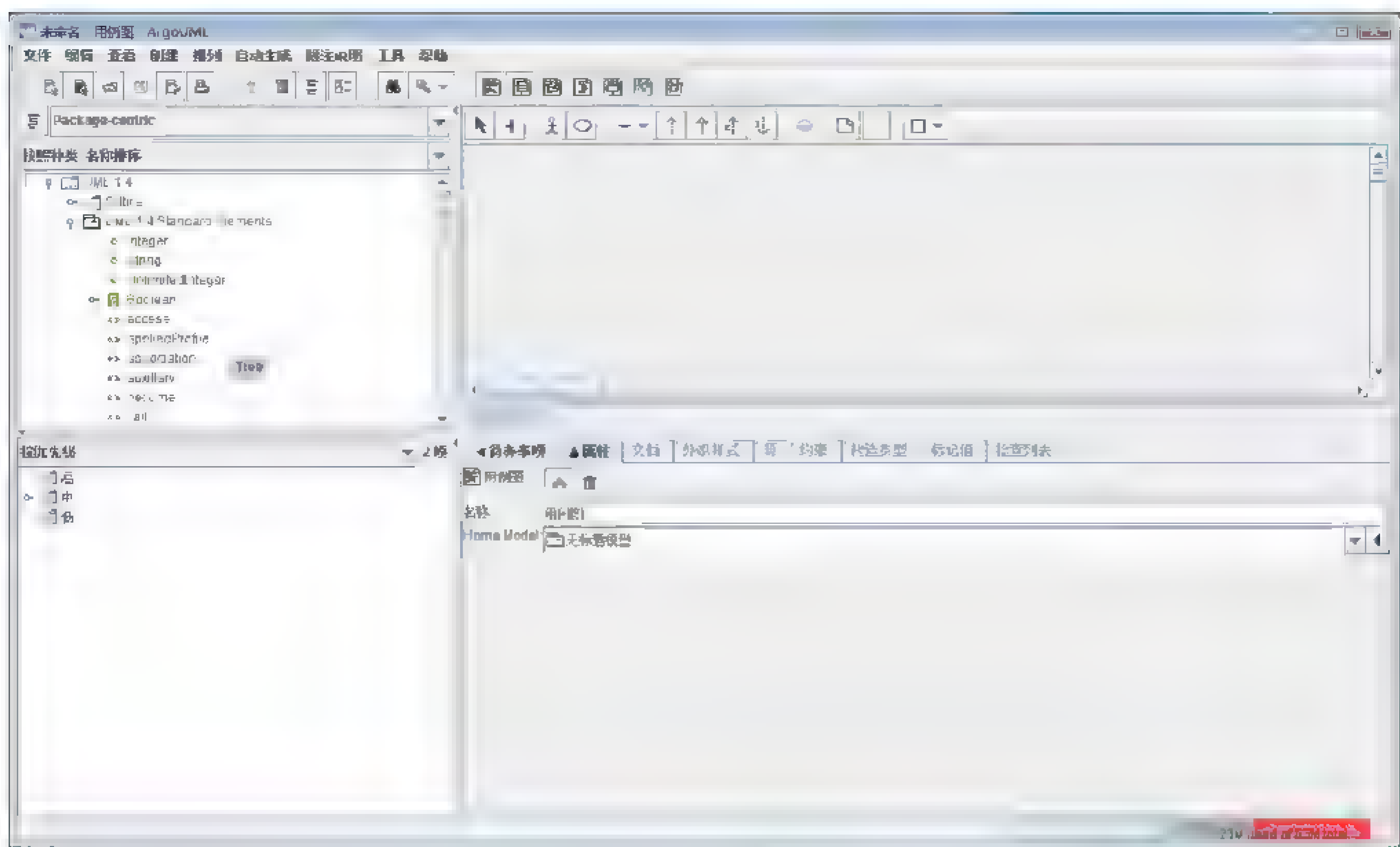

图 3-6　ArgoUML 运行图

从图 3-6 中易知 ArgoUML 只支持 UML 1.4 的九种图示，即是一种基于 Java 的开源 UML OO 建模工具，Argo 是古希腊英雄 Jason 的战船。ArgoUML 支持软件设计者的认知需求，广泛地支持开放标准，如 UML、XMI、SVG（Scalable Vector Graphics，可缩放矢量图形）、OCL（Object Constraint Language，对象约束语言）等；它紧密支持 UML 标准，与平台无关，但建议使用 Java1.5+；无须下载安装，支持 JWS，从浏览器启动运行；可以多种格式导出 UML 图：GIF，PNG，PS，EPS，PGML 以及 SVG；支持包括中文在内的 10 种语言；支持正向工程（支持生成 C++ and C#, Java, PHP4, PHP5, Python, Ruby 代码）和逆向工程（导入 jar 包）；在图中支持数据类型(DataTypes)，构造型(Stereotypes)和枚举(Enumerations)；支持 CallStates, ObjectFlowStates 且兼容 AndroMDA；质量——数百个 bug 得到修正且当前多数功能支持元素多选；支持从浏览树到图的拖拽操作，拖拽操作也适用于在浏览树内操作等。

表 3-2 所示是除 ArgoUML 以外，其他 UML 可以导出 SVG 格式的 UML 建模工具以及可用于 UML 建模的 SVG 编辑工具。

表 3-2　支持 SVG 格式的 UML 建模工具及可制作 UML 的 SVG 编辑工具（IBM）

序号	名称	功能
1	ArgoUML	ArgoUML 一种基于 Java 的开源 UML OO 建模工具，它支持软件设计者的认知需求，广泛地支持开放标准，如 UML、XMI、SVG、OCL 等
2	Batik 1.1 SVG Toolkit	Apache Batik 工具包提供 JAVA 组件创建（SVGGraphics2D）、浏览（JSVGCanvas）和转换（Transcoder）SVG
3	CatWalk	SchemaSoft 的软件工具，用于快速实时创建 SVG Web 应用。在向网站请求数据时，每次都会重新发布数据变化。可以用来实时更新 UML 图
4	Dia	一种基于 GTK+的制图工具，很像 Visio（微软的可视化建模工具）。有一些特殊对象可以帮助绘制实体关系图、UML 图、流程图、网络图，等等，可以将图以 EPS 和 SVG 格式输出

续表

序号	名称	功能
5	DoME（Domain Modelling Environment）	一种元 case 系统，用于构建面向对象软件模型（CY OOA 和 UML），有自己的后端图形语言
6	Gill	即 Gnome Illustration app，是基于 Gnome 的一种通用矢量绘图工具，本身并没有对 UML 提供过多的支持，最终会支持所有的 SVG 特性
7	Gmodeler	一个免费在线 UML 绘图和文档工具，使用 FlashMX 开发，并不支持输出 SVG 格式，但可作为 SVG UML 建模软件的原型参考
8	Graphviz	ATT 出版的开源绘图软件，有 Linux 和 Windows 版本，包括一个名为 Webdot 的 web 服务接口
9	JSeq	可以自动创建 UML 序列图的工具，可输出格式 Zargo 和 SVG。可独立使用或与 JUnit 一起使用
10	MagicDraw UML	非常强大的建模工具，基于 JAVA 开发，可以输出 SVG 格式文件
11	OptimalJ	用于 NetBeans 的一种 UML 类图编辑器，使用 Batik 输出 SVG
12	Poseidon for UML	基于 ArgoUML，与其界面基本相同，完全由 Java 实现，非开源的 UML 建模工具。与 ArgoUML 相比，功能要更丰富，更稳定
13	SVG Maker	一个独立的软件组件，可以作为系统的一部分进行布署
14	SVG Slide Toolkit	它可以把 XML 文件转化为 SVG 幻灯格式，不过用起来似乎有些慢
15	Together Control Center 5.5	经常使用的一种集成化开发平台，使用 Batik 输出 SVG 格式的 UML 图
16	Visual Paradigm for UML Community Edition	支持所有 UML 图，可作为图形输出 SVG、JPG 和 PNG 等格式，执行复杂图的打印。支持从事件流生成序列图，从序列图生成组合图的功能
17	WebDraw	JASC，也就是开发 Paint Shop Pro 的那家公司，提供的一个商业 SVG 可视编辑器

3.2.3.2 UML 2.0 Profile 释义

而对于 UML 2.0 Profile 目的是为描述服务提供一个共同语言，该 profile 包括了在开发生命周期内的很多活动并且为不同的涉众提供了视图[①]。表 3-3 列出了 UML 2.0 元模型在 UML profile 中用作原型元类的元素。

表 3-3 UML 2.0 元模型在 UML profile 中用作原型元类的元素

序号	UML 2.0 元素	原型
1	类	消息，服务划分，服务提供者
2	分类器	服务消费者
3	协作	服务协作
4	连接器	服务信道
5	接口	服务规约说明
6	端口	服务，服务网关
7	属性	消息附件

图 3-7 是一个 UML 2.0 profile 图，它表明了 profile 的实际细节，使用扩展标记（实心的箭头）显示了每一个原型以及它的元类；也可以在该模型中，看到一些约束，尤其是那些在 profile 元素之间的相互约束。

① http://www.ibm.com/developerworks/cn/rational/419_soa/

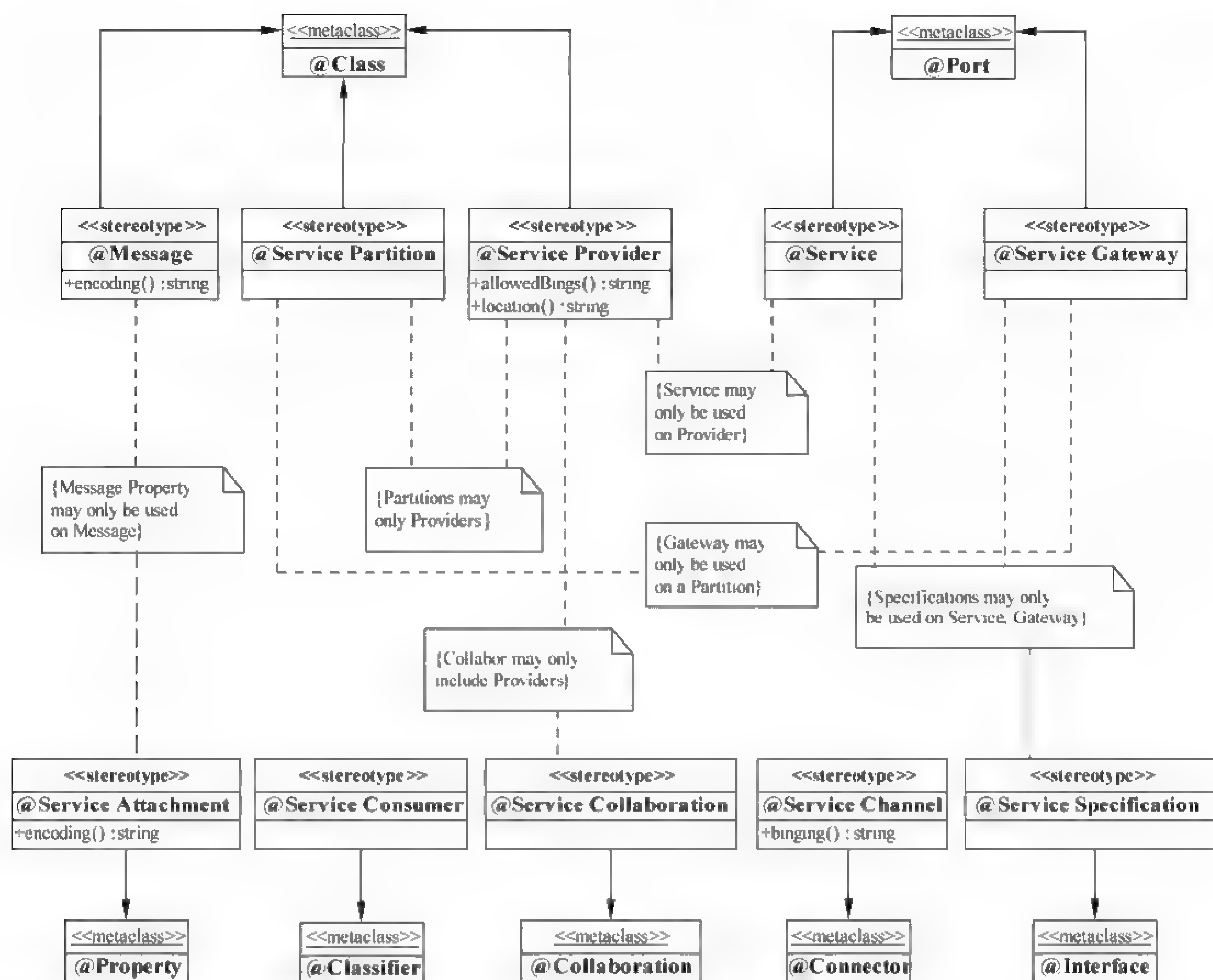

图 3-7　一个 UML 2.0 profile 图

在图中，根据 UML 2.0 profile 描述服务的语义特征，列出了各类原型服务的所表达的语义含义。

1. 信息原型类语义

一个消息代表在 Web 服务描述语言（WSDL）规范中定义的一个概念，即它是实际数据对服务和消费者有意义一个容器。消息不能有操作，但是它可能有属性和与其他类的关联（很可能是某个领域模型），一个消息原型具有一个属性来表示它的假定的编码格式。

2. 原型消息附件的属性语义

这个原型用于表示消息的某些组件是该消息的附件（相对于消息的直接部分本身）。总体而言，不太可能在高级的设计活动中经常使用这个原型，但是对于许多过程，区分附件数据和内嵌的消息数据还是很重要的。例如，一个目录服务可能返回概括的细节作为一个结构化的消息的一部分，但是可以返回图像作为该消息的附件。

3. 原型服务端口语义

该服务模型的元素提供服务交互作用（在 Web 服务技术中）的一个终点，然而这些交互作用的定义是服务规约说明原型的一部分。在实际的模型中，服务不仅识别提供的接口，而且识别还需要的接口（例如回调接口）。同时，一个服务拥有附加的属性表示需要使用的绑定，例如 SOAP-HTTP, SOAP-JMS 等。

4．原型服务信道连接器语义

一个信道表示两个服务之间的通信路径。交互作用也有可能出现在一个信道，但是信道并不表示任何特殊的交互作用。在Web服务世界里，每个服务表示与之相联系的绑定（这样一个客户可以访问它）。在建模一个profile的时候，要么在服务之间的通信，要么在服务与客户之间的通信来表示绑定。

5．原型服务协作语义

服务协作是一种被指定一个服务的实现，并作为一个其他服务的协作的方式。从Web服务的角度来看，这对应于在指定服务实现过程中使用BPEL4WS（Web服务的业务过程执行语言）。所以，这意味着使用一种服务协作作为服务的行为，如果有意要产生一种类似BPEL的语言，它就可能由其他的实现相关的特殊的约束。

6．原型服务消费者分类器语义

任何分类器（如类，组件）可能充当服务的消费者，也包括其他的服务。

7．原型服务网关端口语义

原型服务网关看起来像一个服务，但是它却只是对划分可用的，而对服务提供者是不可用的。一个网关充当一个代理服务，也可以使用它来协调协议或者表示一个划分上的可以利用的接口。

8．原型服务划分类语义

一个划分代表系统的某个逻辑或者物理的边界，建模划分是可选的但是有用的。一个划分可能只具有表示嵌套部分的属性，成为服务或者其他的划分。并且一个划分的符号也比较严格的。

9．原型服务提供者类语义

服务提供者是一个能够提供一个或者多个服务的软件元素。在建模的术语里，可能最经常期望见到的一个UML组件。然而，这样的一个约束似乎难以满足需要，所以为了更大的灵活性，元类被指明作为一个类。

10．原型服务规约说明接口语义

对接口的应用能够表示服务提供的一个操作集合，但通常一个服务可能实现多个接口。一般地，可以将一个协议状态机或者UML 2.0协作附加到服务规约的说明上，使其来表示服务规约说明上的操作调用的顺序。有了这样一个行为的规约说明，任何实现的服务不仅是静态的，而且是动态结构和行为的规约说明，都可以是有效的。当然，服务规约说明只能提供公开的操作，而且每个操作应当只能消耗至多一条消息、产生至多一条消息。

3.2.3.3 UML基本方法

前面回顾了UML的背景及各个版本的情况，以及以UML 2.0为基础，指出了UML 2.0较前版本的所新增和增强的功能和UML 2.0 Profile的释义。下面就从UML的最终的MDD、MDA和UML组成进行概述。

1．模型驱动开发（MDD）

模型驱动开发（Model-Driven Development，MDD）是软件开发的一种样式，其中主要的软件工件是模型，并可以从这些模型生成代码和其他工件。模型是从特定角度对系统进行的描述，它省略了相关的细节，因此可以更清楚地看到感兴趣的特性（例如，结构工程师会

创建适合于确定建筑物承载特性的模型）。并且对象管理组（OMG）给 MDD 概念贴上了模型驱动的体系架构（Model Driven Architecture，MDA）的标签，开发了一组标准来支持 MDD，而在 MDD 中的过程是从软件开发需求阶段早期的业务逻辑定义开始。而且在对业务逻辑进行抽象的基础上，可使用统一建模语言（UML）对业务逻辑建模，从而能够得到的一个或多个模型，使其形成生成代码并获得具体需求实现的基础。但模型驱动方法不是一种独特地新式方法，在过去就已经被使用并获得了不同程度地成功。近十年来对 MDD 愈来愈受重视的原因是其支持性技术已经愈来愈成熟，而成熟点在于比起过去的情况来看在实践上更加自动化。这不仅仅在效率方面，而且在可测量性方面也是这样。同样，MDD 具有的能力是与继承性工具和方法相结合，使得使用 MDD 相关的工具时更加便利舒适，给使用者带来显而易见的好处。

MDD 主要具有加快开发过程，业务逻辑从平台中独立出来（如果业务逻辑变化，模型也要变化），降低软件开发的成本等多种优势。且可以用多种形式表示，如 UML、XML Model Interchange、Essential Meta Object Facility 和 W3C XML Schema。同时可以在开源集成 Eclipse 中引入 MDD 插件进行相应的开发，其中之一就是 Eclipse Modeling Framework（EMF），它是 Eclipse 中进行模型驱动开发的基础。

EMF 是 Eclipse Open Source Project 的一个工具子项目，它是一种建模和数据集成框架，也是一个用于为 Eclipse 创建插件的代码生成框架；它使用 ECore 元语言描述模型，并为这些模型提供运行时支持。所使用的 ECore 元语言是以 OMG Meta Object Facility 2.0（MOF）的一个子集 Essential MOF（EMOF）为基础来描述模型。且 EMF 模型被持久化为 XML Model Interchange（XMI）文档。EMF 也提供了查看和基于命令编辑模型的能力，以及按照 EMF 模型操纵和序列化实例文档的基本编辑器，EMF 模型可从带注释的 Java 代码、XML 文档或 UML 模型产生。

（1）MDD 作为示意图和蓝图的模型。目前，利用模型来设计软件是一个公认的实践，通常模型大多用于传达系统某个方面的示意图，或用于手动实现的详细设计的蓝图。并且将模型作为文档和规范是有价值的，但是这需要严格的规程来确保模型与实现进度保持一致。

（2）MDD 用于模型自动生成。在 MDD 中，模型不仅用作示意图或蓝图，还作为生成有效实现的主要工件。并且在 MDD 中，当开发新的软件组件时，首要的焦点是面向应用领域的模型。当然，代码和其他目标领域的工件是利用建模专家和目标领域专家的参与设计，并由转换规则来生成。同时，MDD 能够极大地减少解决方案开发的成本，并且提高解决方案的一致性和质量。这些优点是通过自动化、带有转换的实现模式来实现的，使得它消除了重复的低层次开发工作。当在构建解决方案工件时，与其重复地手动应用技术经验，还不如将这些经验技巧直接编入转换中，并且还能带来一致性和可维护性的优势。

最终使得 MDD 将应用程序开发的重点从平台上移开，让具有应用程序开发经验的开发人员在领域平台级概念的情况下，不用关注具有平台经验的开发人员的详细情况来设计应用程序。平台经验直接在转换中获取，而不是编制为项目指导，或者在项目进行的过程中重新去多次发现。同样，关于实现架构的决策也直接编入转换中，而不是编制为架构决策的文档。

（3）MDD 具有自动化特性。虽然代码和其他平台工件的生成是 MDD 的重要部分，但是 MDD 样式的自动化有更深的意义。软件开发项目需要生成许多非代码的工件，其中一些完全或部分地来源于模型。

（4）为 MDD 项目估计任务。MDD 对构建应用软件的方式有着深远的影响，它获得了技术人员的经验及决策，并通过为项目的需求所定制的工具，来使得余下的团队可以获得经验和决策。由于大量的低层次编码工作已经自动化了，所以开发的成本，以及测试业务软件的成本极大地减少了；错误的数量也减少了，并且在工作完成的方式上增加了一致性。然而，当 MDD 需要管理一个项目中的另一个项目时，内部工程包含了 MDD 工具的开发，这些工具可以供开发团队在外部项目中构建业务应用程序时使用。一般来说，当开始确定要将一个业务应用程序利用 MDD 工具构建时，这时就需要着重于需求，并且可以对该方法进行一些调整来适合另一个项目。若所调整的项目一旦开始开发了，就可以将 MDD 工具用于构建许多业务应用程序。同时，当对于两个不项目，谨慎地组织和计划是非常重要的，特别是在一开始的时候，由于在与开发项目相关的平常问题之上，存在着管理额外的内部依赖集的需求。这时，MDD 工具需求必须在应用程序开发人员需要它们之前进行确认和开发。当两个项目的任务流要互相联结时，这样就可以确保由 MDD 工具来交付产品，从而实现不同类型用户的需求。图 3-8 所示展示了 MDD 项目中的任务。

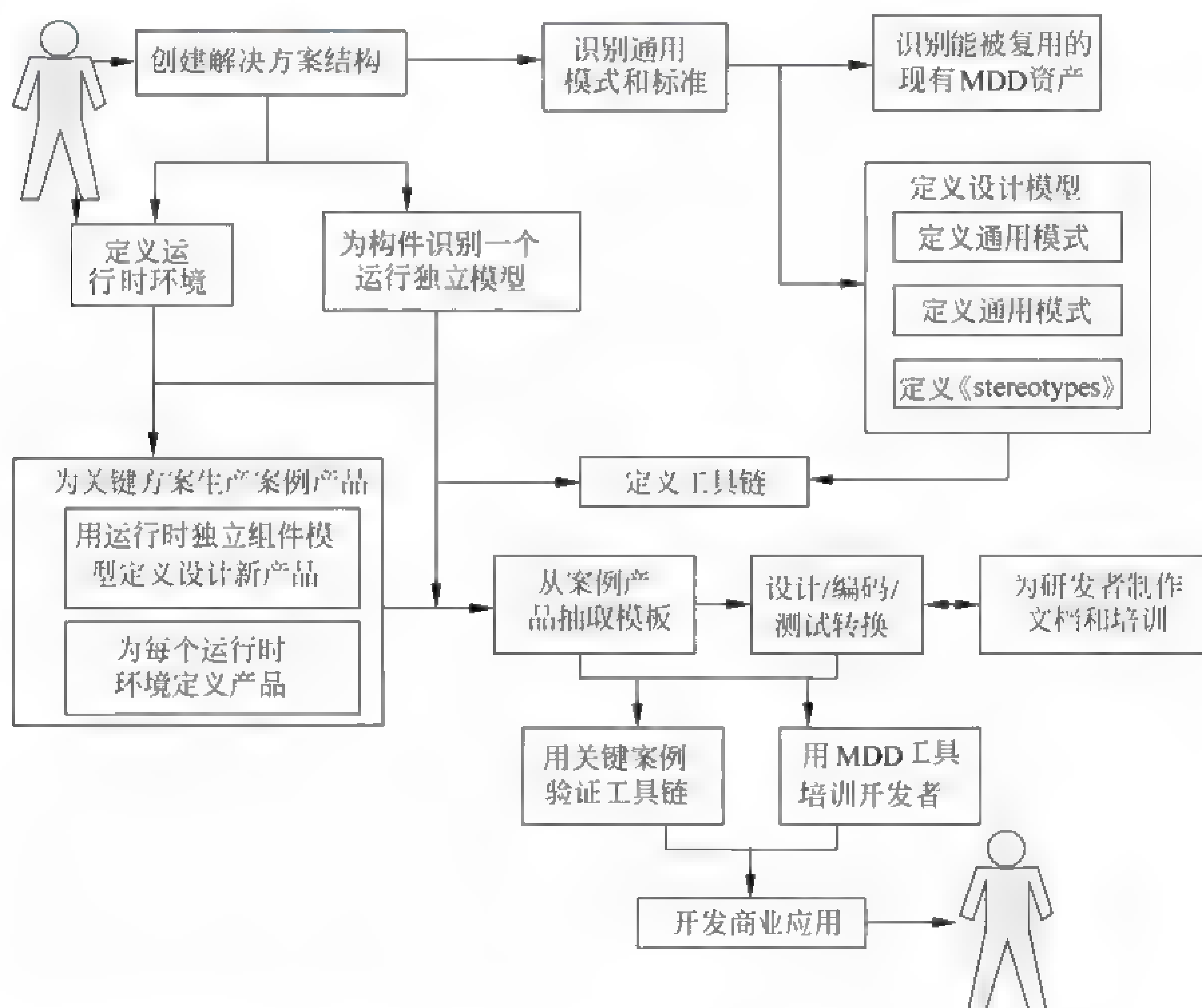

图 3-8　MDD 项目中的任务流

（5）MDD 开发。图 3-9 中的流程展示了开发人员如何利用 MDD 工具来开发部分业务应用程序。在此实例中，开发人员审阅了业务问题，并且选择了设计模式。该模式部分地填充了设计模型，而开发人员填写他们正在构建的具体业务功能的细节，此后，开发过程就完全自动化了。当开发人员选择一项来生成工件量时，这些工件被打包并放入构建区。然后开发人员可以选择更多选项，为个别的运行时平台生成附加的工件。

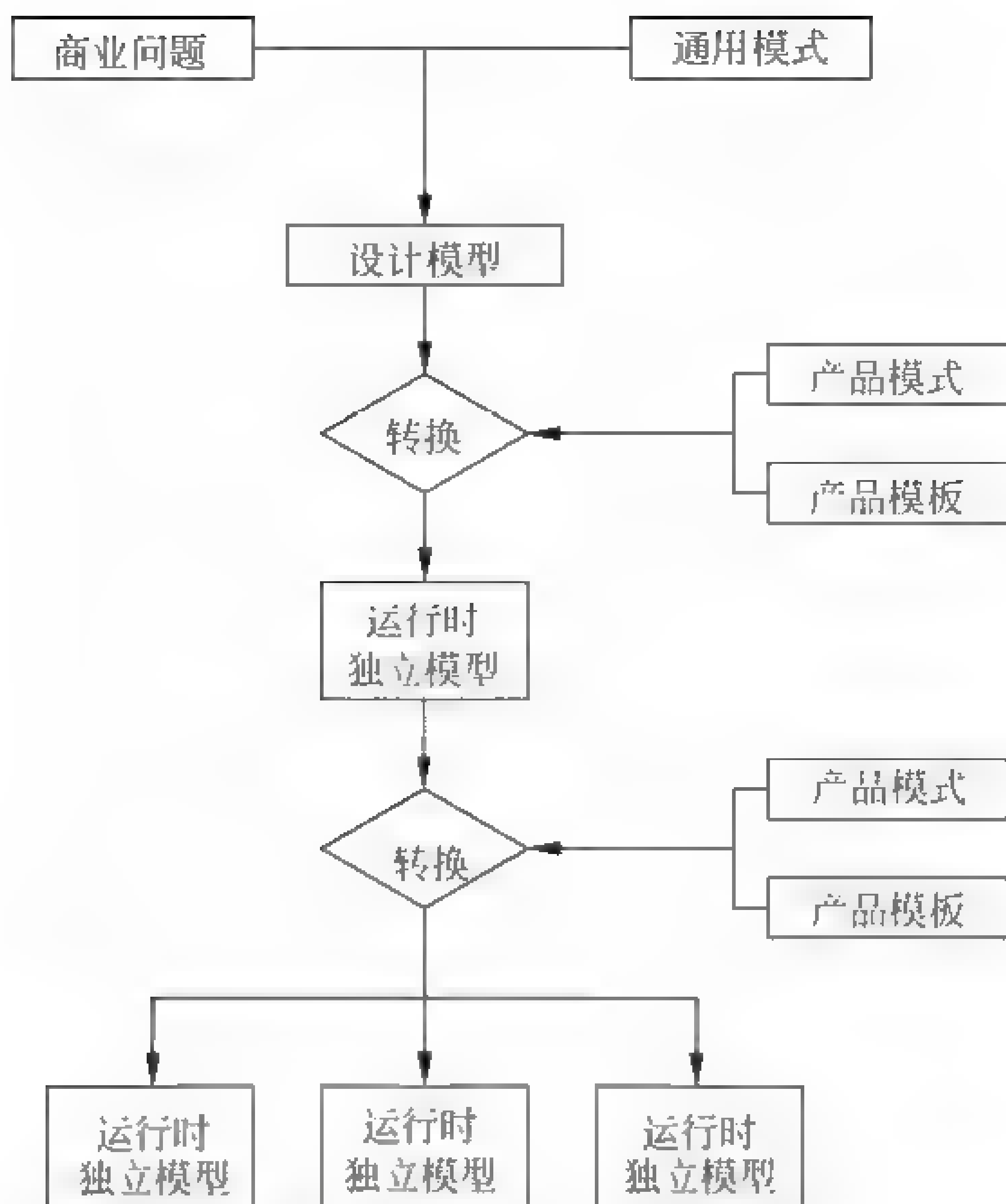

图 3-9　MDD 工具开发

（6）MDD 的优点。表 3-4 所示是 MDD 优点总结。

表 3-4　MDD 的优点

序号	优点	描述
1	增加了生产力	通过由模型生成代码和工件的方式，减少了软件开发的成本，同时增加了开发人员的生产力
2	可维护性	（1）许多解决方案组件是使用遗留平台技术实现的，但组织不再掌握这方面的技术了。MDD 形成了可维护的架构，在其中可以快速一致地做出变更，可以将组件更有效地移植到新技术上
		（2）高层次的模型与不相关的实现细节无关，这使得处理底层平台技术及其技术架构中的变更更加容易。可以通过更新转换来变更实现的技术架构。转换也被重复地应用于原有的模型，用于生成依据新方法的实现工件
		（3）可以在做出最终决策之前尝试不同的想法。使不好的决策很容易变更。但人们经常按照错误的决策继续进行软件项目，而成本过高难于修复
3	遗留系统的复用	通常在 UML 中为现有的遗留平台建模。如果有许多组件是在同一个遗留平台上实现的，那么可以开发从组件到 UML 的逆向转换。也可以将组件移植到新的平台上，或者生成包装器（wrapper），通过集成技术（例如 Web 服务）来访问遗留组件
4	适应性	由于已经在自动化方面进行了投资，因此添加或修改业务功能就是很简单的了。当添加新的业务功能时，只需要开发针对该功能的行为。并且生成实现工件所需的剩余信息就可以在转换中获取
5	一致性	手动地应用编码实践以及架构决策是容易出错的。MDD 确保一致地生成工件
6	可重复性	当在程序或组织层应用时，MDD 尤其强大，来自开发转换的投资回报在每次复用时都有所增加。使用经过试验和测试的转换还增加了开发新功能的可预测性，并且减少了风险，因为架构和技术问题已经解决了
7	改进了涉众的交流	模型省略了与了解系统逻辑行为无关的实现细节。它们更接近于问题领域，减少了涉众所了解的概念，与表示解决方案所用语言之间的语义差异。简化了与业务目标更好地结合的解决方案的交付

续表

序号	优点	描述
8	改进了设计的交流	模型帮助在设计层上了解系统，引出了对系统的改进的讨论和交流。因为模型是系统定义的一部分，比起文档，它们从不会过时，而且是可靠的
9	经验获取	项目或组织经常依靠重复地做出最佳实践决策的重要专家。他们的经验可以在模式和转换中获得，因此，他们不需要直接面对项目的其他成员。而且，有了伴随转换的充足文档，即使当专家离开时，组织的经验仍旧保留在模式和转换中
10	模型可以作为长期的资产	模型是获取组织的 IT 系统的功能的重要资产。高层的模型对最新平台级上的变更具有弹性。它们只在业务需求变更时才发生变更
11	推迟技术决策的能力	早期的应用程序开发针对建模活动。可以推迟具体技术平台或产品版本的选择，直到有更多的信息可用时再选择。在出现极其长的开发循环（例如，航空交通管制系统）的领域中，这是至关重要的。在开发开始时，目标平台可能还不存在

UML 2.0 为了符合模型驱动架构（Model Driven Architecture）的需求也做了大幅度的修改，除在图形基础上扩充及变化了部分的展现方式外；也增加了一些图形标准元件，并比前一版多出了由顺序图与互动图所混合而成的互动概图（Interaction Overview Diagram），强调时间点的时序图（Timing Diagram）与合成结构图（Composite Structure Diagram），此外，在UML 2.0 中，UML1.0 合作图转变为通信图（Communication Diagram），且在顺序图中也添加了互动框（Interaction Frame）的概念，还有增加一些运算子（如 sd、loop、alt 等）。同时，UML 2.0 支援模型驱动架构（MDA）倡议，提供稳定的基础架构，容许软件开发程序增添自动化作业。此外，MDA 把大型的系统分解成几个元件模型，并与其他模型保持连接，使得UML 更加精确。

2. 模型驱动架构（MDA）

型驱动架构（Model Driven Architecture，MDA）是由 OMG 定义的一个软件开发框架，是一种基于 UML 以及其他工业标准的框架，支持软件设计和模型的可视化、存储和交换。和 UML 相比，MDA 能够创建出机器可读和高度抽象的模型，这些模型独立于实现技术，并以标准化的方式储存。MDA 把建模语言用作一种编程语言，不仅仅是设计语言，而是 MDA 的关键之处是模型在软件开发中扮演了非常重要的角色。同时，MDA 是属于 OMG 支持的悠久传统和过去二十年中的众多计算机标准。OMG 一直负责对于一些系统说明和互操作性方面上的、工业上知名的和最具影响力的规范开发，包括公共对象请求代理体系架构（CORBA），OMG 接口定义语言（IDL），Internet Inter-ORB Protocol（IIOP），统一建模语言（UML），Meta Object Facility（MOF），XML Metadata Interchange（XMI），Common Warehouse Model（CWM）和 Object Management Architecture（OMA）。此外，OMG 也增强了这些规范来支持特定行业，比如卫生保健业、制造业、电信业和其他的行业。图 3-10 所示是 MDA 的应用领域，从图中易知，这个体系结构的核心建立在 UML、MOF 和 CWM 上。

MDA 把系统操作的规范描述从系统利用底层平台能力的方式以细节形式分离出来。MDA 提供了一种途径（通过相关的工具）来规范化一个平台独立的系统和规范化平台，从而为系统选择一个特定的实现平台，并且把系统规范转换到特定的实现平台。MDA 的主要目标是：通过架构性的分离来实现轻便性、互操作性和可重用性。并且 OMG 正在将 MDA 作为一种开发更加准确的、满足客户需要的、在系统的演进中具有更好灵活性的系统方法来

促进它的发展。MDA 方法构建在较早期的系统规范标准的工作上，并且为定义相互连接的系统提供一种全面的具有互操作能力的框架。下面从以下几个方面来描述 MDA 的形态[①]。

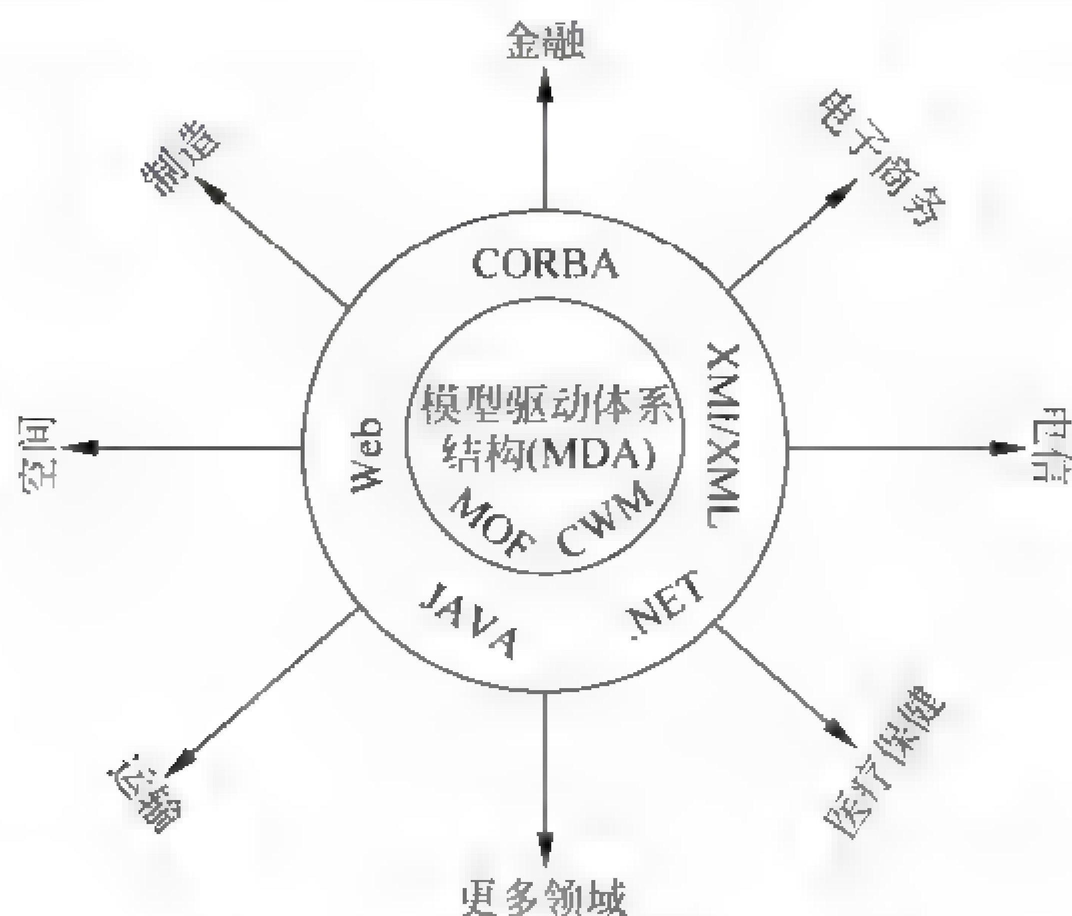

图 3-10　OMG 模型驱动体系结构

（1）MDA 具有中间件特性和处理遗留系统的能力。由于 MDA 在核心上是以平台形式独立的，就使得向这个互操作环境中添加新的中间件平台非常简单。即在确定新平台表示和实现公共中间件概念和功能的方式之后，OMG 成员可以把这些信息作为映射合并到 MDA 中。各种面向消息的中间件工具，再加上 XML/SOAP（简单对象访问协议）和.NET，都将以这种方式集成进来。事实上，通过消除某些行业提出的 XML 文档类型定义（DTD）的冲突，MDA 可以帮助企业对不同的文档互操作。随着多种中间件平台添加到 MDA 中，MDA 越来越成熟，类似于桥接器（bridge）、网关（gateway）和从一种平台到另一种平台的映射等集成工具的生成将变得更加自动化。

同样，遗留应用程序是目前阻碍软件发展的另一个挑战，很多应用程序是在组件环境还没有出现之前构建的，不能很好地适合任何一种核心模型。但是，通过包装与 MDA 核心模型一致的代码层，遗留应用程序也可以引入到 MDA 中。如果首先构建包装程序的 MDA 模型，那么包装程序的外部部分（即面向网络并与其他应用程序和服务互操作的那一部分）就可以自动生成，至少可以部分自动生成。包装程序的另一部分（即对遗留应用程序本身进行调用并返回的那一部分）通常必须手工编码。

（2）MDA 具备 Internet ORB。MDA 是目前正在开发的下一代 OMG 标准，它可以作为 ORB（Object Request Broker，对象请求代理程序）来整合所有中间件平台。OMG 是对 ORB 理解最深刻的组织，最适合于肩负扩展这一概念的职责。同时，不仅仅是中间件标准，而是成为一种中间件中立的、模型驱动的方法，为用户带来以下具体好处：

① 企业可以使用所选的中间件构建新的基于 MDA 的应用程序。因为应用程序的基本语义系统凝聚在平台独立的模型中，如果将来需要使用不同的中间件（或者相同中间件的新版本），这种迁移是相当易于管理的。此外，通过使用一致的体系结构和某种程度的代码生成，能够系统地建立与企业中其他基于 MDA 的应用程序之间互操作性的桥梁和通路，以及与客户、供应商、业务合作伙伴之间的联系。

① http://www.ibm.com/developerworks/cn/rational/rationaledge/content/mar05/403/index.html

② 保持公司业务正常运转的遗留应用程序，只要对遗留系统相关接口和功能包装并将其功能合并到 MDA 中，就可以与企业当前应用程序互操作。这些程序仍然保留在建立它们的平台上。因此，MDA 可以帮助自动构造从一个平台到另一个平台的桥梁。

③ 各个层次的行业标准将包括按照 MDA 核心标准定义的平台独立模型，即可以获得完成标准功能的标准设施，而不必自己构建，由于这些标准设施均以 MDA 为基础，因此提高了其互操作性和可演进性。

④ 当出现新的中间件平台时，OMG 的快速且遵循多数意见的标准化过程将通过定义新的标准化来映射，并把它们合并到 MDA 中。然后，MDA 工具将针对新的平台实现到平台独立模型的转换。

⑤ 开发人员可以获得最大限度的灵活性，即当底层基础设施随时间变化时，能够从稳定的、平台独立的模型重新生成代码。在软件生命期尤其是长期的支持和维护过程中（这也是应用程序生命周期中最昂贵的一个阶段），应用程序和领域模型的重用将提高 ROI（Return On Investment）。

⑥ 模型通过 UML 来构建、查看和操纵，并通过 XMI 传输，然后在 MOF 资料库中存储。

⑦ 系统语义的正式文档（通过建模）将提高软件质量，延长设计的有效生命周期（从而提高 ROI）。

（3）MDA 具有标准化领域模型。自从 1996 年 1 月以来，很多 OMG 成员参加了 Domain Task Forces（领域任务组，DTF），使得标准化、特定纵向型市场的服务和设施成为了社区关注的焦点。目前，这些规范包括用 OMG 交互式数据语言（Interactive Data Language，IDL）编写的接口和相应的英文语义说明。因此 OMG 的领域规范就属于这种情况，因为它们的模型没有从 IDL 接口中单独表示出来。由于它们的模型是隐含的，这些服务和设施在对于 CORBA 环境之外，既没有得到它们应得的认可，也没有得到广泛的实现和使用，尤其是考虑到它们的底层模型的质量，把隐含的模型扩展到 CORBA 外显然是合理的。而且基本上每个 DTF 都会产生所属应用程序空间中用于标准设施的标准框架。其中包括规范的、平台独立的 UML 模型，并附有非正式的、特定于平台的 UML 模型以及至少一种目标平台的接口定义。MDA 的公共基础也会促进部分代码的生成和实现，当然这些代码不是标准化的。

（4）MDA 具有普及服务功能。OMG 在 UML 的平台独立模型层上定义了普及服务（Pervasive Services）。只有在确定了服务的特征或者体系结构之后，才能够为 MDA 支持的所有中间件平台生成特定于平台的定义。在平台独立业务组件模型的抽象层中，服务在非常高的层次上进行描述（类似于组件开发者在 CCM 或 EJB 中的视图）。当模型映射到特定平台时，开发人员将使用所选的开发工具来生成调用该平台上的本地服务代码（或者动态调用它）。普及服务仅对底层应用程序可见，即那些直接为服务编写的应用程序。同样，硬件和软件属性（如可伸缩性、实时性、容错性或者嵌入式特征）也要进行建模，通过定义这些属性的 UML 表示，或者定义把属性与企业计算结合起来的环境容错性情况，OMG 将对 MDA 进行扩展，以便支持和集成带有这些期望特征的应用程序。图 3-11 是 MDA 支持普及服务的结构图。

（5）MDA 使用原则。OMG 组织对于 MDA 的使用有以下四个原则：

① 以定义良好的符号表示的模型是理解企业级方案系统的基础；

② 系统的构建能够围绕着一系列模型，通过使用在模型之间的一系列转换来组织的，并

且能被组织到一个分层的和转换的体系架构框架中；

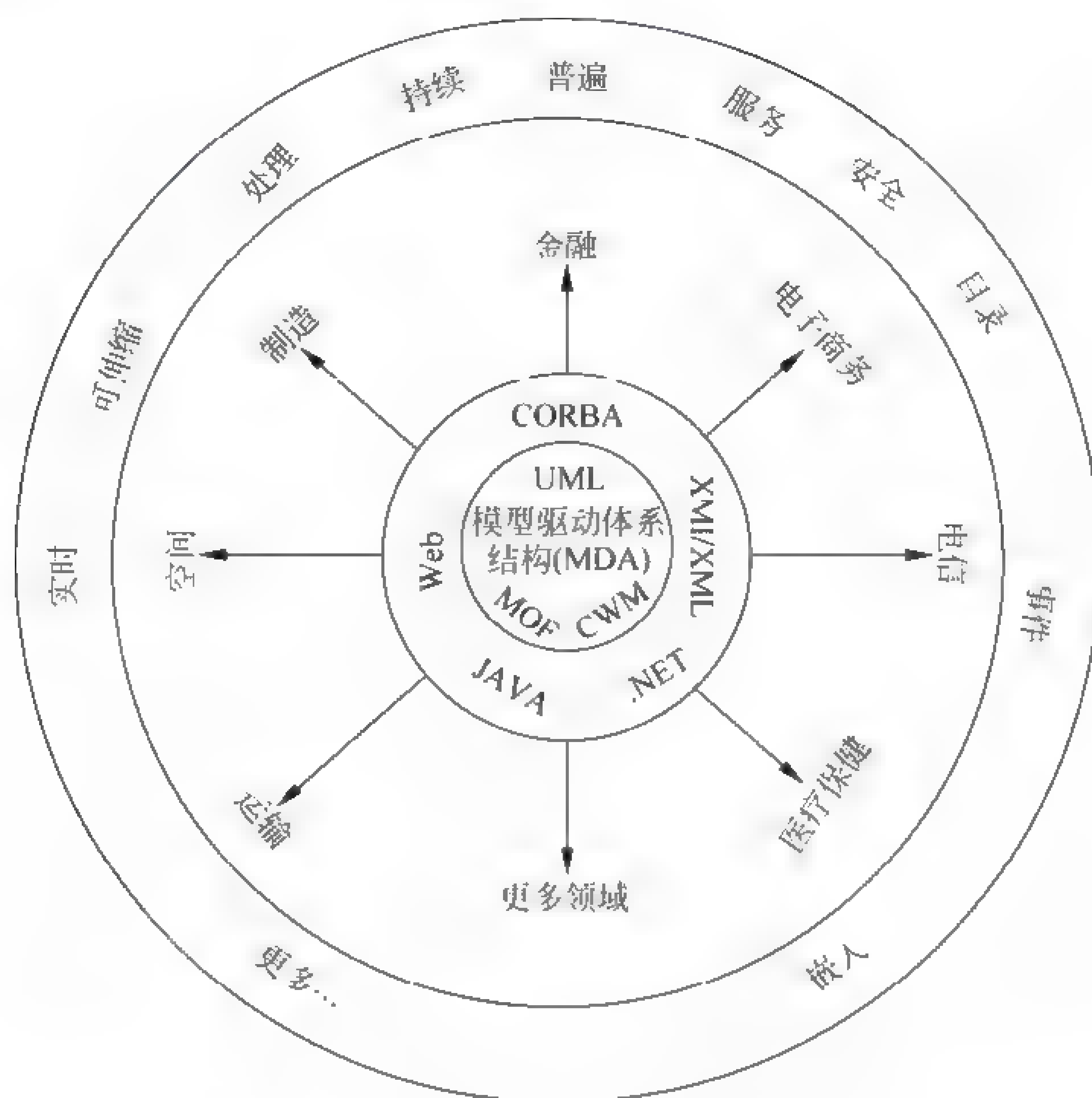

图 3-11　MDA 支持普及服务和专门计算环境

③ 以一系列元模型来描述模型的一种正式的、支持能够使在模型中有意义的集成和转换变得容易，并且是通过工具来实现自动化的基础；

④ 接受和广泛采纳基于模型的方法需要工业的标准来提供开放性给客户，并鼓励供应商之间的竞争。为了支持这些原则，OMG 已经定义了一系列的层次和转换，从而为 MDA 提供了概念性的框架和词汇表。特别的，OMG 确定了四种模型类型：计算无关的模型（CIM），平台无关的模型（PIM），平台模型（PM）描述平台的相关模型（PSM）和一个实现相关的模型（ISM）。图 3-12 所示是在 MDA 下 PIM 与 PSM 的映射关系图，图 3-13 所示是在 PSM 下 UML Profile 与 WSDL 的图。

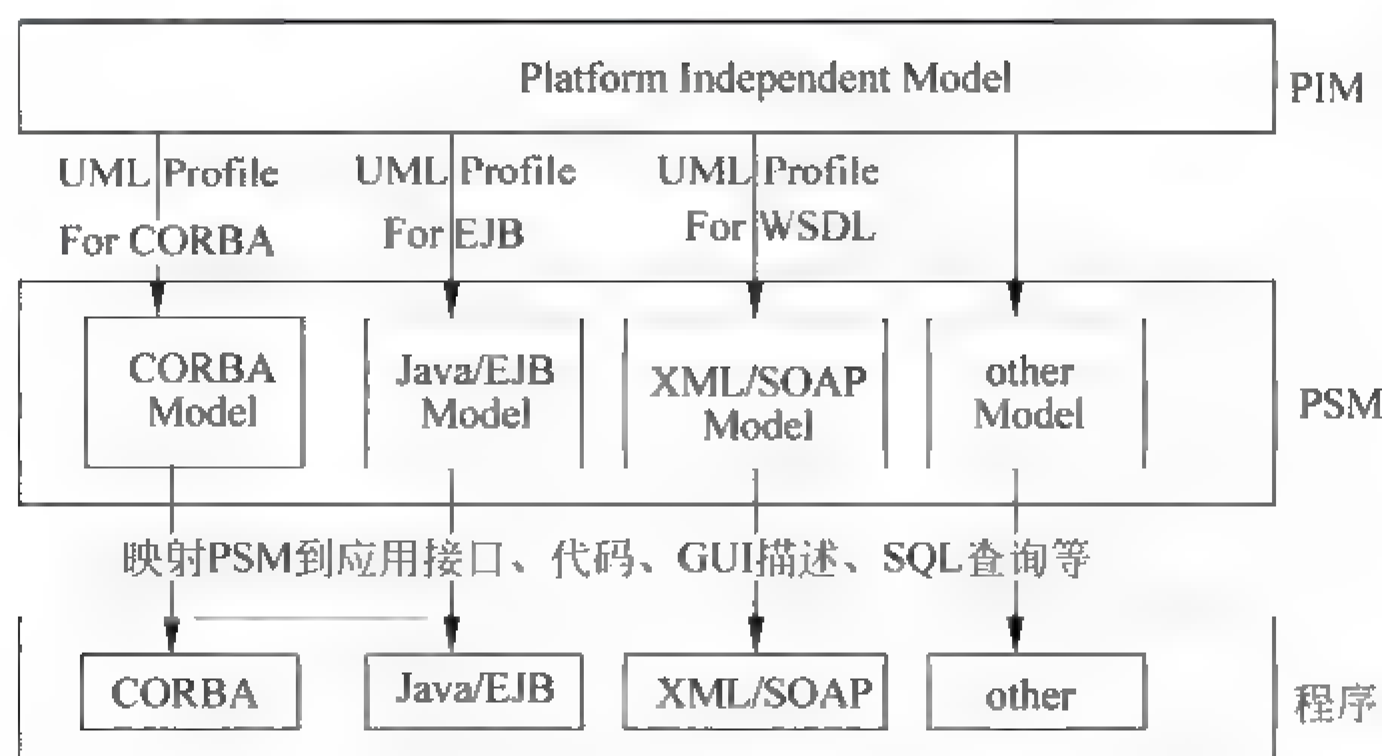

图 3-12　在 MDA 下 PIM 与 PSM 的映射关系图

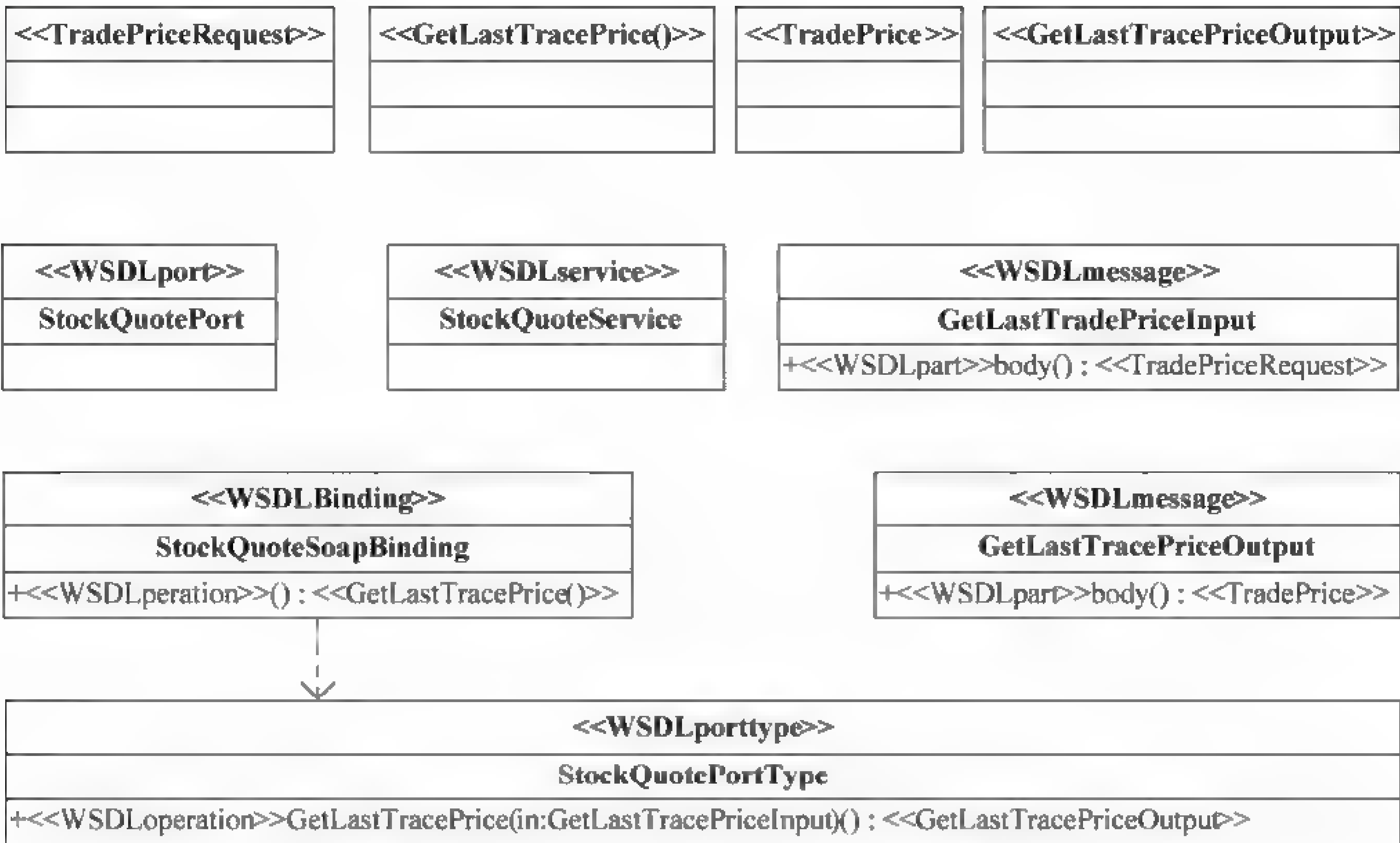

图 3-13　PSM 下 UML Profile 与 WSDL 的图

（6）MDA 下的模型转化。MDA 技术的核心概念均是 OMG 的一系列标准：UML、OCL、MOF、XMI（XML-based metadata Interchange，基于 XML 的元数据交换）、CWM（Common Warehouse Metamodel，公共仓库元模型），其中元对象设施（Meta Object Facility，MOF）是 MDA 的核心部分。MDA 下的每一层模型都基于一个特定的元模型，即对表述模型的语言进行定义的模型，而所有的元模型又都基于 MOF。元模型是用来定义转换规则的重要元素，通过使用 MOF 结构定义，可以定义在建模语言之间的转换规则。这样源模型与目标模型的转换可以通过在基于 MOF 的源元模型与目标元模型之间定义转换规则来完成，如图 3-14 所示，并在源元模型与目标元模型之间定义一套对应语义的转换规则，这套转换规则必须使得两模型包含相同的语义内容。在源模型与目标模型之间通过专门的模型转换引擎来应用转换规则，完成从元模型到目标模型之间的转换。在图 3-14 中的源元模型是 UML 活动建模语言，目标元模型是 BPEL 语言。

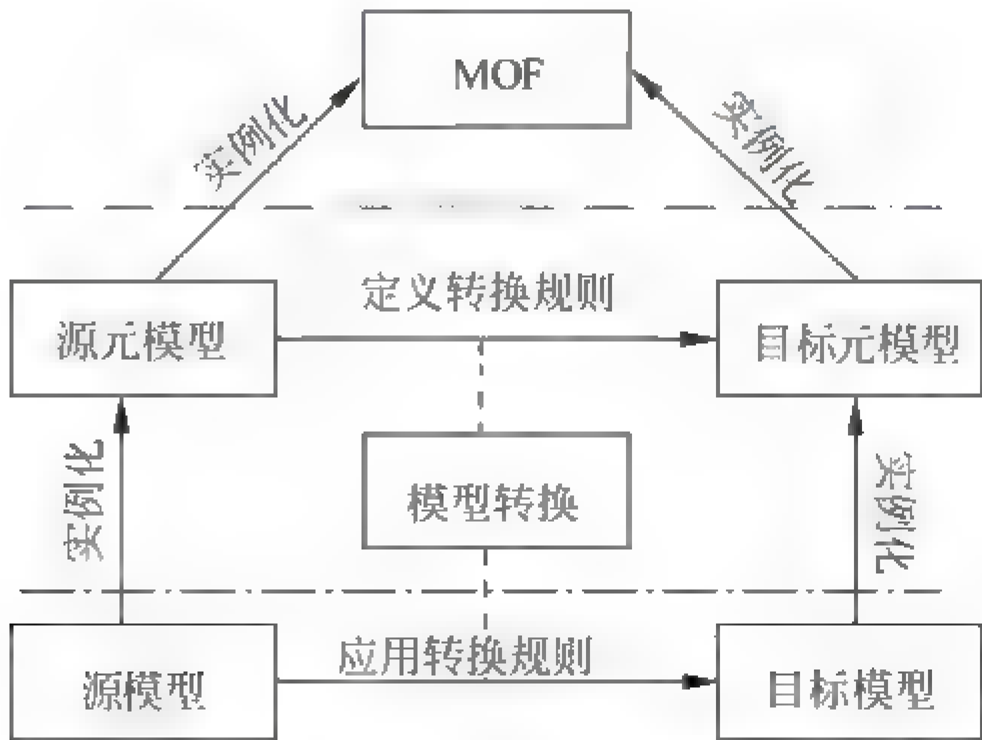

图 3-14　MDA 下的模型转化

（7）MDA 软件开发生命周期。MDA 生命周期和传统生命周期没有什么区别，主要的区

别在于开发过程创建的工件，包括 PIM（Platform Independent Model，平台无关模型）、PSM（Platform specific Model，平台相关模型）和代码。PIM 是具有高抽象层次、独立任何实现技术的模型。PIM 被转换为一个或多个 PSM，它还是为某种特定实现技术量身定做，如图 3-15 所示。MDA 的出现，为提高软件开发效率，增强软件的可移植性、协同工作能力和可维护性，以及文档编制的便利性指明了解决方法，也使得 UML 的用途走得更远。

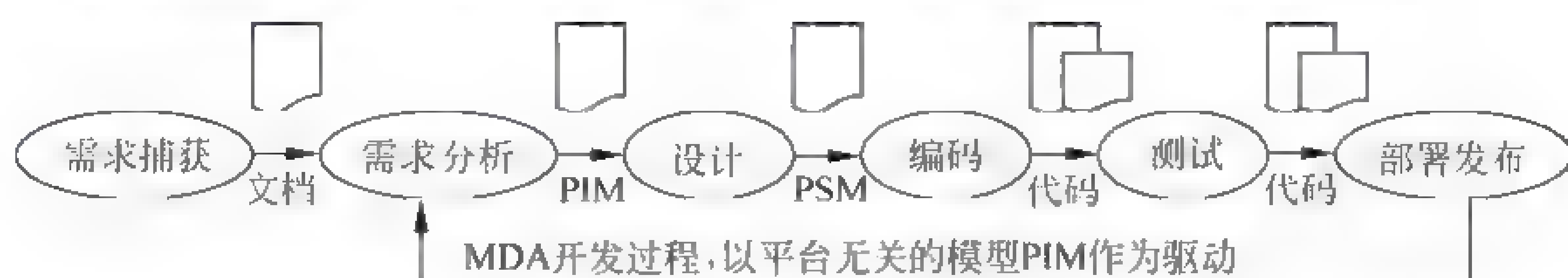

图 3-15　MDA 软件开发生命周期

（8）常见 MDA 开源实现平台。对于 MDA 的实现来说，现在有两种不同的途径：一种是使用模型驱动来实现应用开发过程；另一种是直接利用模型驱动运行时的应用系统的行为方式。前一种实现，大部分支持从 UML 模型转换的代码生成工具，例如 AndroMDA，而后一种实现就比较少，例如 openMDX。

① AndroMDA。AndroMDA（http://www.andromda.org）是一种遵循模型驱动结构（MDA）范例的代码生成框架。它从 CASE 工具中获得的一个 UML 模型，并生成一个完全可部署的应用程序，主要用途在于从 UML 模型生成 Hibernate，EJB，Spring，WebServices 和 Struts 等框架标准所对应的代码，并且在开发过程的建模阶段可以快速生成可运行原型，就此而言它是非常实用且有效的工具，但是它的代价就是增加了很多对应各种框架类的 stereotype，这样的模型事实上已不能再算作 PIM 了，这样不利于平台的迁移和模型的复用。而 openMDX 则仅仅使用了两个用于语义描述的 stereotype，这样的模型显得更加中立，更面向业务建模的视角。在 Struts 和 Spring 已经成为事实上的 J2EE 框架标准的情况下，AndroMDA 能够满足很多 J2EE 项目的框架要求，并且节约了很多重复性的编码工时。

但是，AndroMDA 的长处也正是它的短处，因为完全面向 J2EE 平台开发，对于通用、中立的类型没有定义，也缺少对于属性的特性支持，比如持久性属性和导出属性的区分。在模型的表达上，仍然是更倾向于从技术框架的角度进行建模和描述系统行为。图 3-16 是 AndroMDA 结构图。

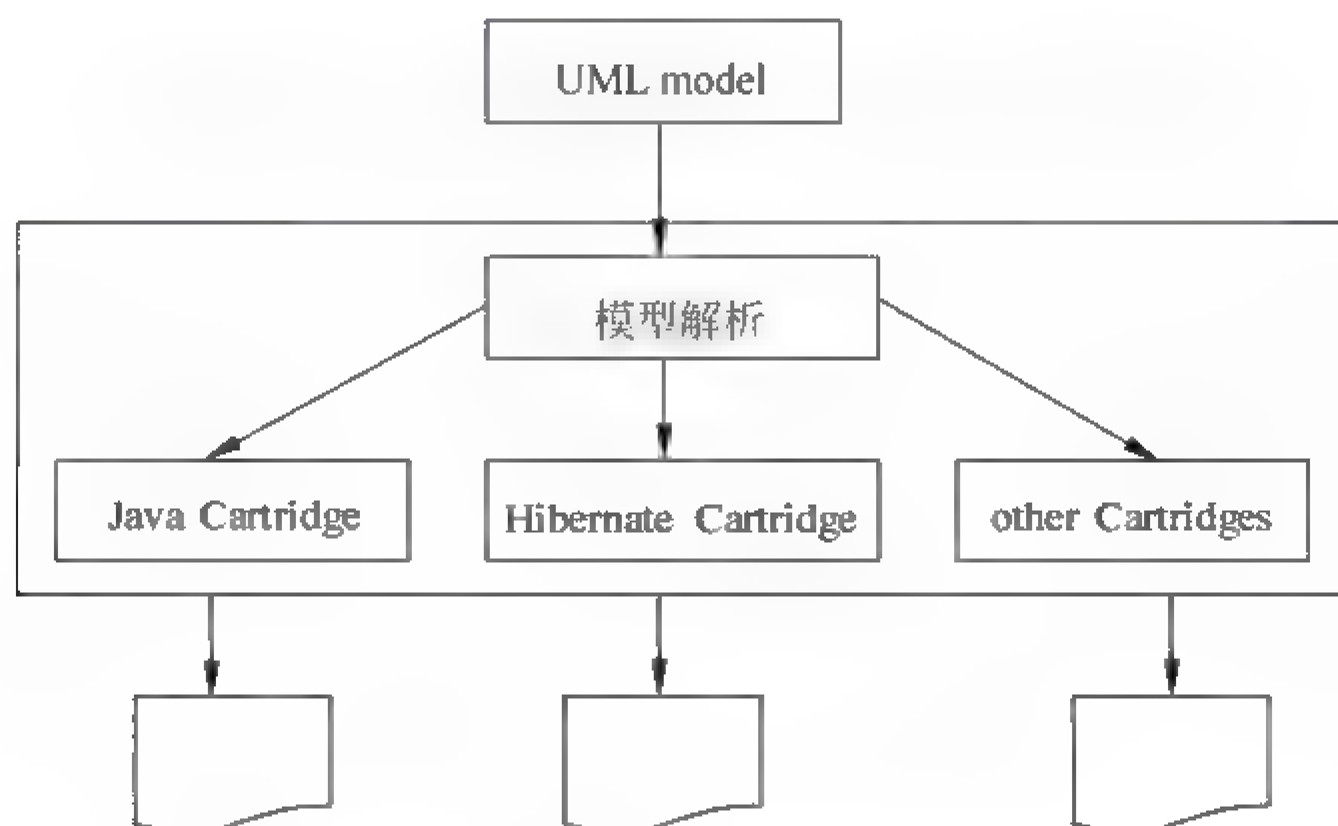

图 3-16　AndroMDA 结构图

② openMDX。openMDX是一个OMG Model Driven Architecture（MDA）为起始的高级实现。它是一个工业化的、开放的、模型驱动的运行时引擎和平台独立模型（PIMs）框架。不像大多数商业工具，openMDX没有实现PIM到PSM映射的方法。而是提供了一个普通的，分布式的对象引擎（作为PIM平台）。

openMDX拥有自己独立的中立性框架，支持J2EE、.Net和CORBA平台等，同时具有极大的灵活性和扩展性。基于openMDX的系统，从桌面应用程序到完全分布式应用的转化，不需要改变一行代码。在openMDX所使用的模型中，也没有采用私有的模型标签和功能描述。openMDX没有提供UML模型工具，也没有提供一个IDE（Integrated Development Environment）来支持整个开发测试和配置应用，但是支持所有主流的UML模型工具，如Rational Rose、MagicDraw、Poseidon、Together等。但现有的插件已经提供了基本功能，如持久性、审核、历史记录和角色以及强大的日志功能，并携带了一个集成的、完整的界面生成引擎。但是缺点也是显见的，由于采用的框架完全围绕模型来运行和部署，迫使开发人员从习惯的开发方式和设计思考模式转变到完全不同的重心上。例如，不再首先考虑UI、业务层、持久层所采用的方式和数据库结构等，而是首先考虑业务模型中的对象及其关联关系，当然这在真正的业务建模中是核心部分。OpenMDX将模型的这种核心地位延伸到了开发、部署和交付使用阶段。图3-17所示是OpenMDX开发流程。

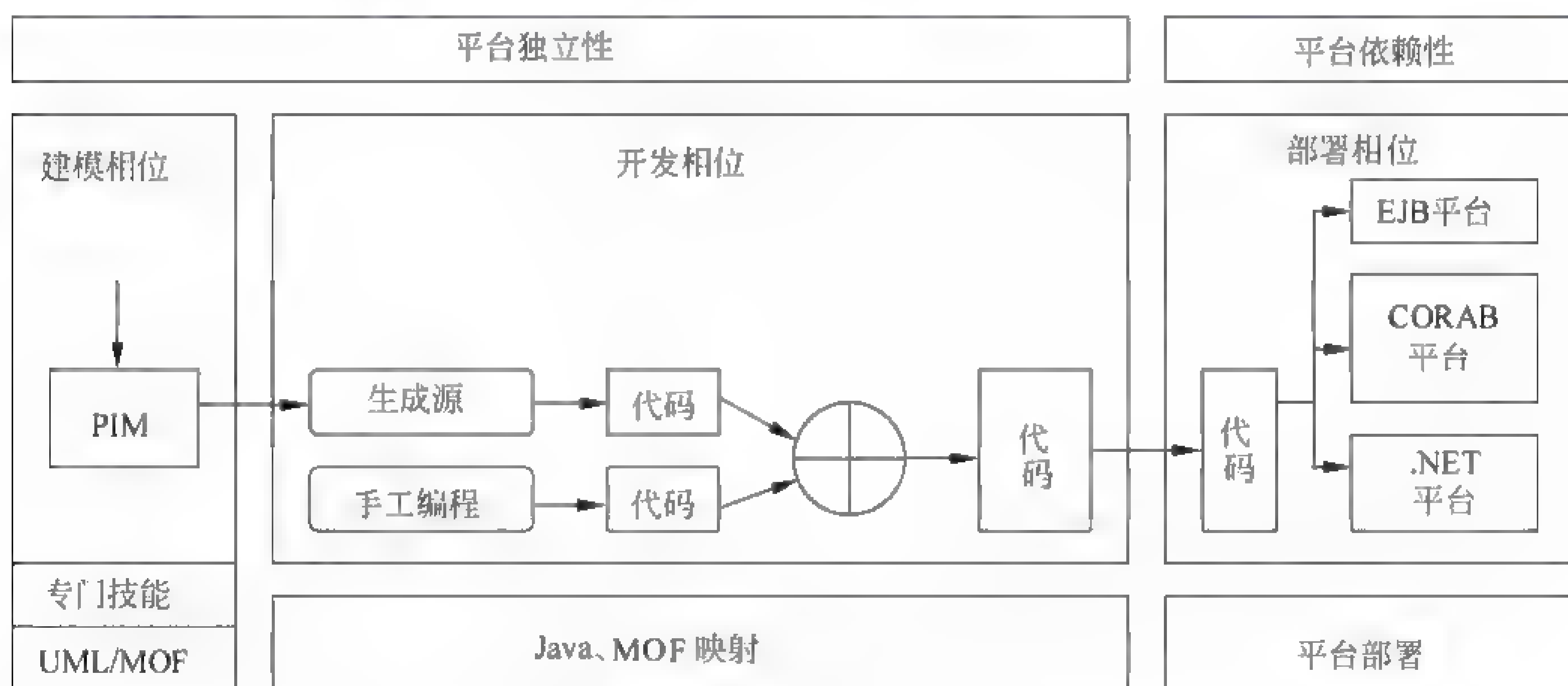

图3-17　OpenMDX开发流程

在图中的各个表示描述如下：

- **Modeling phase（建模相位）** 创建PIM，openMDX提供了工具将兼容MOF的模型映射，即采用了JMI和XMI的映射，并由MOF映射所定义。类图表达了一个组件对外的行为规范，例如组件的接口。
- **Development phase（开发相位）** 编写代码。
- **Deployment phase（部署相位）** 将生成的MOF映射，手工编写的代码，部署信息和openMDX运行时环境，在部署阶段集成在一个可运行、可部署的应用服务平台上，例如，J2EE应用服务器、.Net服务器等。

（9）MDA优缺点。表3-5是MDA优缺点的总结。

表 3-5　MDA 优缺点

序号	优点	缺点
1	对建模的投资将更加持久、有效	在为一个应用建立 PIM 的时候，基本上没有技术上的周旋空间
2	具有了技术上的灵活性	软件开发的创造性在一定程度上减弱了
3	将不再受技术或应用所具有的不同变化周期的影响	潜在的不成熟性，UML 2.0 还在幼年时代，MDA 工具出现的时间也相对很短，这说明还隐藏了很多风险

下面是一个 openMDX 的简单应用例子（即常用的 HellowWord 的例子），图 3-18 所示是用 UML 图表示这个 HellowWord[①]。

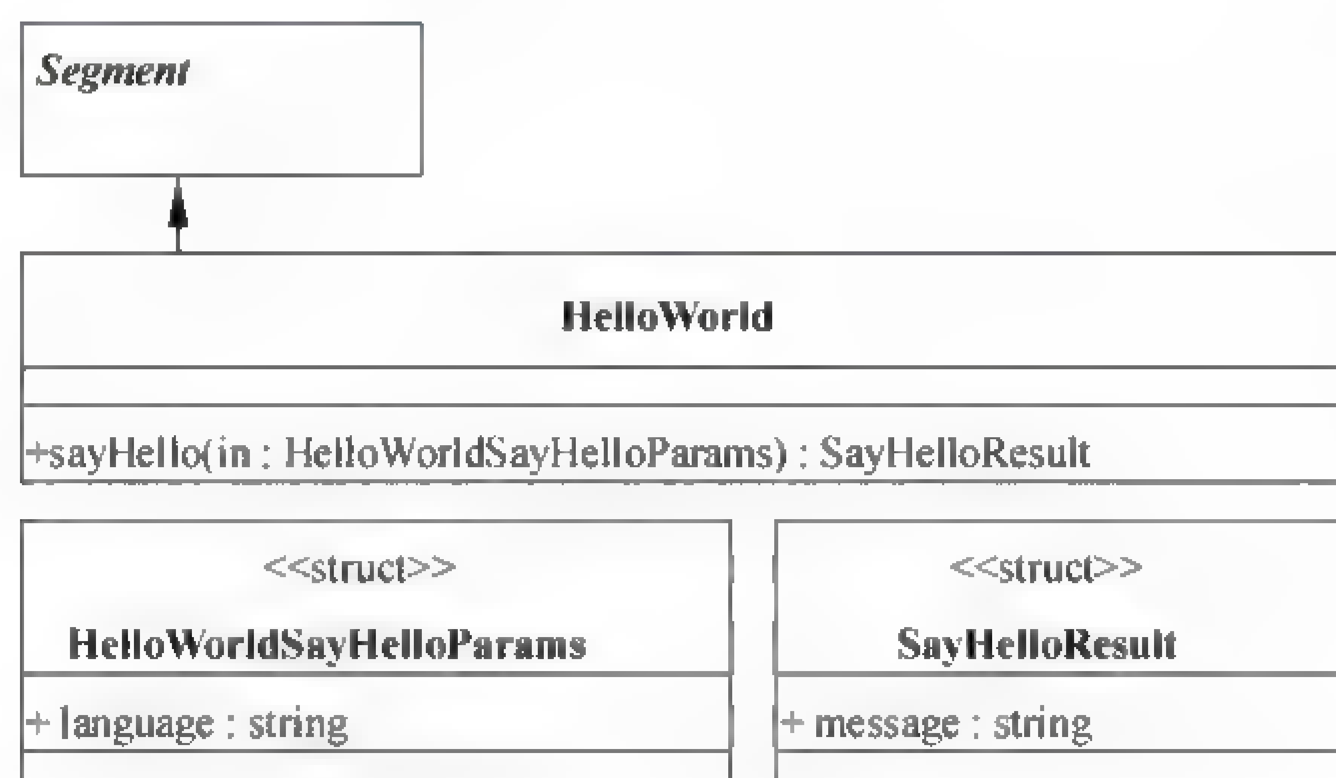

图 3-18　基于 openMDX 的 HellowWord 例子

如下程序是 hello world 客户端调用 sayHello()的 openMDX：

```
import org.openmdx.example.helloworld1.*;
javax.jdo.PersistenceManagerFactory pmf =
JDOHelper.getPersistenceManagerFactory("EntityManagerFactory");
javax.jdo.PersistenceManager pm = pmf.getPersistenceManager("guest", "guest");
HelloWorld helloWorld = (HelloWorld)pm.getObjectById(HELLOWORLD XRI);
HelloWorldPackage helloWorldPkg =
(HelloWorldPackage)helloWorld.refOutermostPackage().refPackage(Helloworld
1Package.class.getName());
pm.currentTransaction().begin();
org.openmdx.example.helloworld1.SayHelloResult hResult = helloWorld.sayHello(
helloWorldPkg.createHelloWorldSayHelloParams("en"));
pm.currentTransaction().commit();
System.out.println("Client: sayHello[en]=" + hResult.getMessage());
```

3. UML 图示表示组成

UML 是实现项目开发流程的一个重要工具，它是一套可视化建模语言，由各种图和模型来表达。图就是用来显示各种模型元素符号的实际图形（通常包括用例图、协作图、活动图、

① http://sourceforge.net/apps/trac/open-mdx/wiki/Introduction

序列图、部署图、构件图、类图、状态图），这些元素经过特定的排列组合来阐明系统的某个特定部分或方面。一般来说，一个系统模型拥有多个不同类型的图，一个图是某个特定视图的一部分。通常，图是被分配给视图来绘制的。另外，根据图中显示的内容，某些图可以是多个不同视图的组成部分。图也是模型中信息的图形表达方式，但是UML模型独立于UML图的存在。因此对UML有以下两种定义：

（1）UML语义：描述基于UML的精确元模型定义，元模型为UML的所有元素在语法和语义上提供了简单、一致、通用的定义性说明，使开发者能在语义上取得一致，消除了因人而异的最佳表达方法所造成的影响，此外UML还支持对元模型的扩展定义。

（2）UML表示法：定义UML符号的表示法，为开发者或开发工具使用这些图形符号和文本语法为系统建模提供了标准，并且这些图形符号和文字所表达的是应用级的模型，在语义上它是UML元模型的实例。

在UML系统开发中有三个主要的模型：

（1）功能模型（从用户的角度展示系统的功能，包括用例图）；

（2）对象模型（采用对象，属性，操作，关联等概念展示系统的结构和基础，包括类图）；

（3）动态模型（展现系统的内部行为，包括序列图，活动图，状态图）。下面以UML 2.2的14种图示来描述UML图，并着重介绍其中九种图示。其实UML使用一套与Java语言或其他面向对象语言等价物，同时也是本体论等价物的图形标记。UML的内涵远不只是这些模型描述图，但是对于初学者来说，这些图对这门语言及其用法背后的基本原理提供了很好的介绍。通过把标准的UML图放进的工作产品中，精通UML的人员就更加容易加入项目并迅速进入角色。最常用的UML图包括：用例图、类图、序列图、状态图、活动图、组件图和部署图。图3-19所示是UML图示结构。

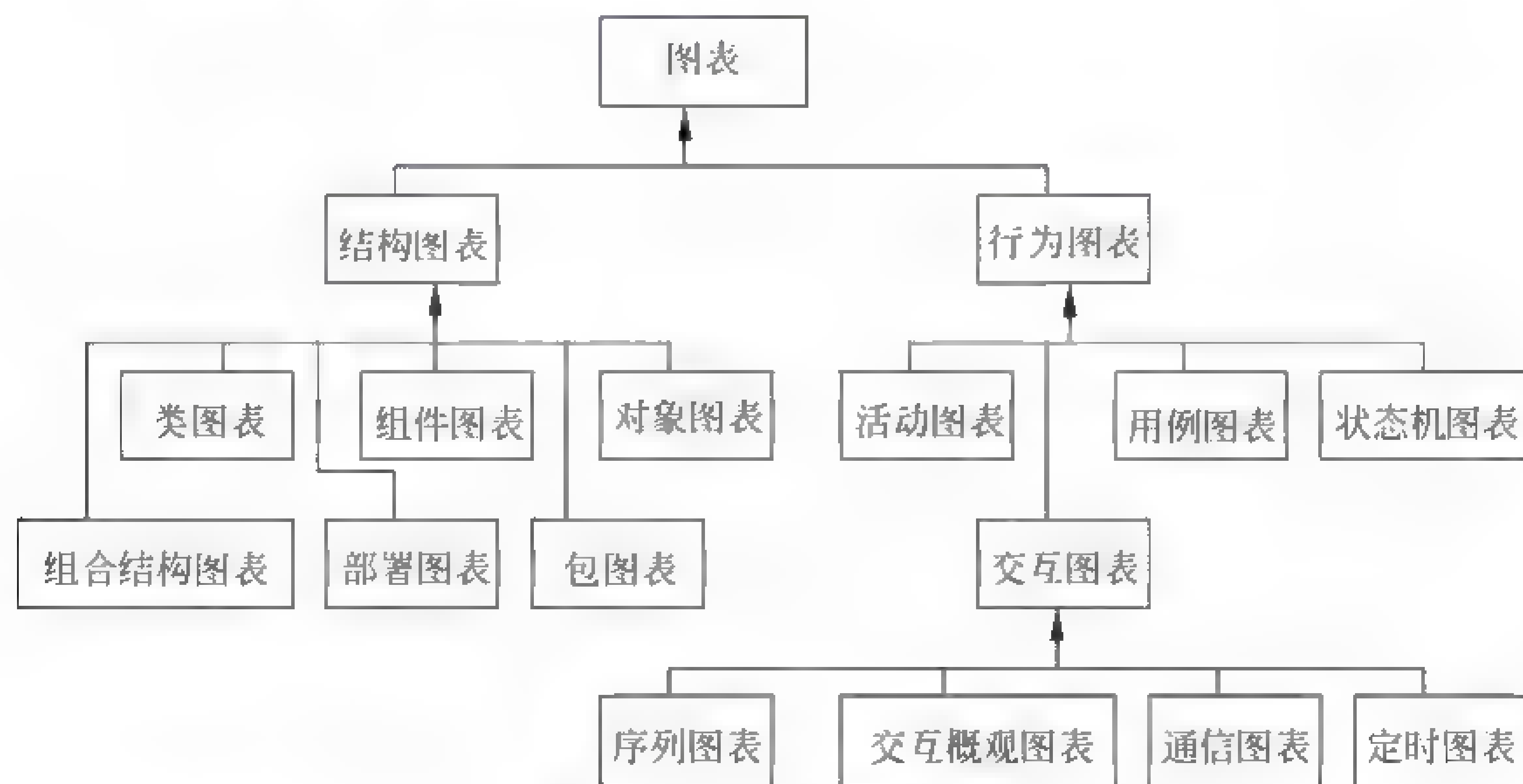

图3-19 UML图示结构

对图3-19的说明如下：

（1）类别图（Class Diagram）。类别图是软件工程的统一建模语言（UML）一种静态结构图，该图描述了系统的类别集合，类别的属性和类别之间的对象建模关系。一般都被用于概念建模（conceptual modelling）的系统分类的应用程序，并可将模型建模转译成程式码。为了进一步描述系统的行为，这些类图可以辅之以状态图或UML状态机。类图用矩形表示，共分三层：最上面是类别名称、中间部分包含类别的属性、底部部分包含类别的方法。各类

图一般具有的关系为：外部链接、关联、聚合、组成、继承、实现、依赖、多重。图 3-20 是常用的 CheckingAccount 和 SavingsAccount 类如何从 BankAccount 类继承而来的类图。

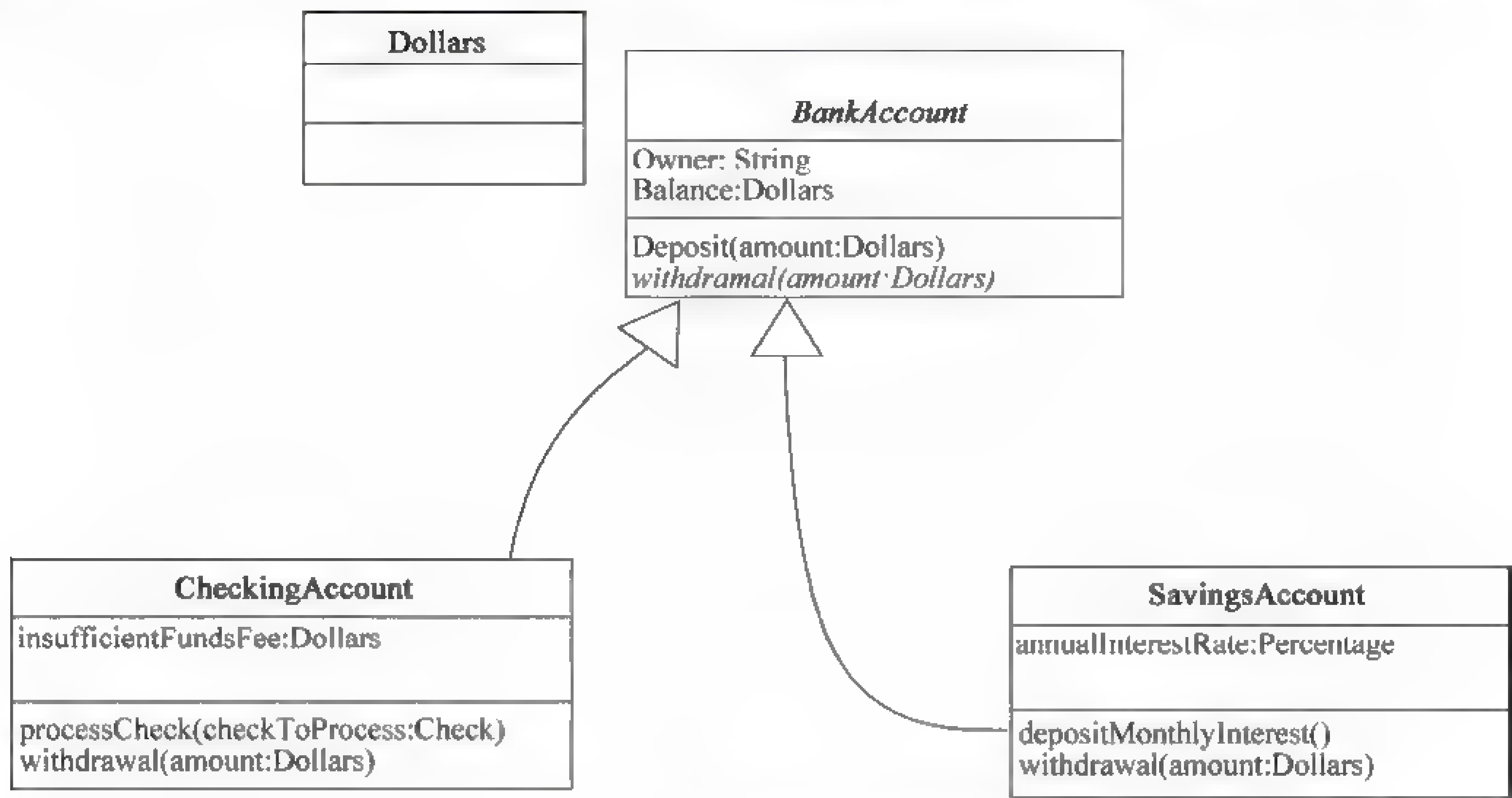

图 3-20　具有继承关系的类图

（2）用例图（Use Case Diagram）。用例（Use Case）是一种描述系统需求的方法，使用用例的方法来描述系统需求的过程就是用例建模。用例方法最早是由 Iva Jackboson 博士提出的，后来被综合到 UML 规范之中，成为一种标准化的需求表述体系。用例图描述了系统提供的一个功能单元，其目的是帮助开发团队以一种可视化的方式理解系统的功能需求。一般表示出用例的组织关系，要么是整个系统的全部用例，要么是完成具有功能的一组用例。

用例方法完全是站在用户的角度上（从系统的外部）来描述系统的功能的，在用例方法中，一般被定义系统看作是一个黑箱，使得并不关心系统内部是如何完成它所提供的功能的。用例方法首先描述了被定义系统有哪些外部使用者（抽象成为 Actor），这些使用者与被定义系统发生交互；针对每一参与者，用例方法又描述了系统为这些参与者提供了什么样的服务（抽象成为 Use Case），或者说系统是如何被这些参与者使用的。所以从用例图中，我们可以得到对于被定义系统的一个总体印象。与传统的功能分解方式相比，用例方法完全是从外部来定义系统的功能，它把需求与设计完全分离开来。在面向对象的分析设计方法中，用例模型主要用于表述系统的功能性需求，系统的设计主要由对象模型来记录表述。另外，用例定义了系统功能的使用环境与上下文，每一个用例描述的是一个完整的系统服务。

使用用例的方法来描述系统的功能需求的过程就是用例建模，用例模型主要包括以下两部分内容：

① 用例图（确定系统中所包含的参与者、用例和两者之间的对应关系，用例图描述的是关于系统功能的一个概述）；

② 用例规约（针对每一个用例都应该有一个用例规约文档与之相对应，该文档描述用例的细节内容）。图 3-21 是常用的一个用例图。

（3）序列图（Sequence Diagram）。是一种 UML 行为图，它通过描述对象之间发送消息的时间顺序显示多个对象之间的动态协作。它可以表示用例的行为顺序，当执行一个用例行为时，时序图中的每条消息对应了一个类操作或状态机中引起转换的触发事件。一般由对象

（Object）、生命线（Lifeline）、消息（Message）和激活（Activation）四部分组成。

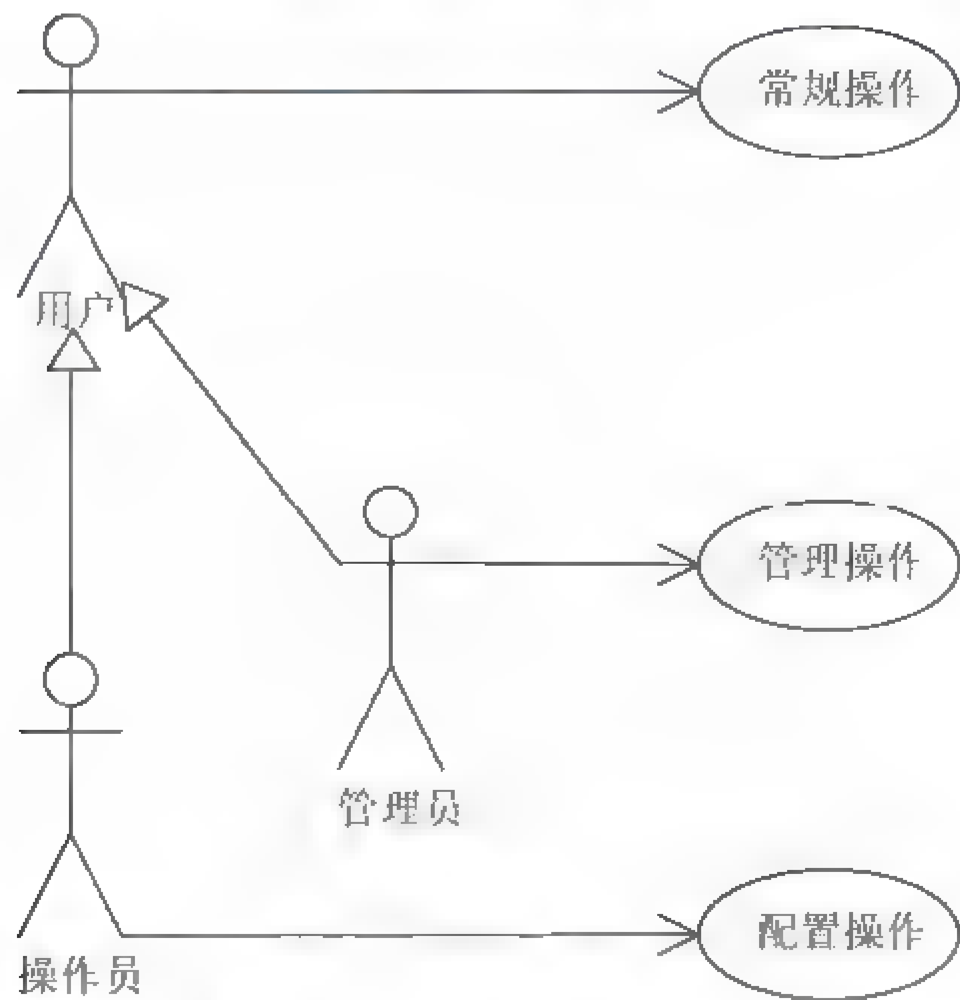

图 3-21　用例图表示方法

序列图主要用于按照交互发生的一系列顺序，显示对象之间的这些交互。然而，一个组织的业务人员会发现，序列图显示不同的业务对象如何交互，对于交流当前业务如何进行很有用。除记录组织的当前事件外，一个业务级的序列图能被当作一个需求文件使用，为实现一个未来系统传递需求。在项目的需求阶段，分析师能通过提供一个更加正式层次的表达，把用例带入下一层次。那种情况下，用例常常被细化为一个或者更多的序列图。

序列图的主要用途之一是把用例表达的需求，转化为进一步、更加正式层次的精细表达。用例常常被细化为一个或者更多的序列图。序列图除了在设计新系统方面的用途外，它们还能用来记录一个存在系统（称为“遗产”）的对象现在如何交互。当把这个系统移交给另一个人或组织时，这个文档很有用。图 3-22 就是一幅关于手机支持平台的序列图。

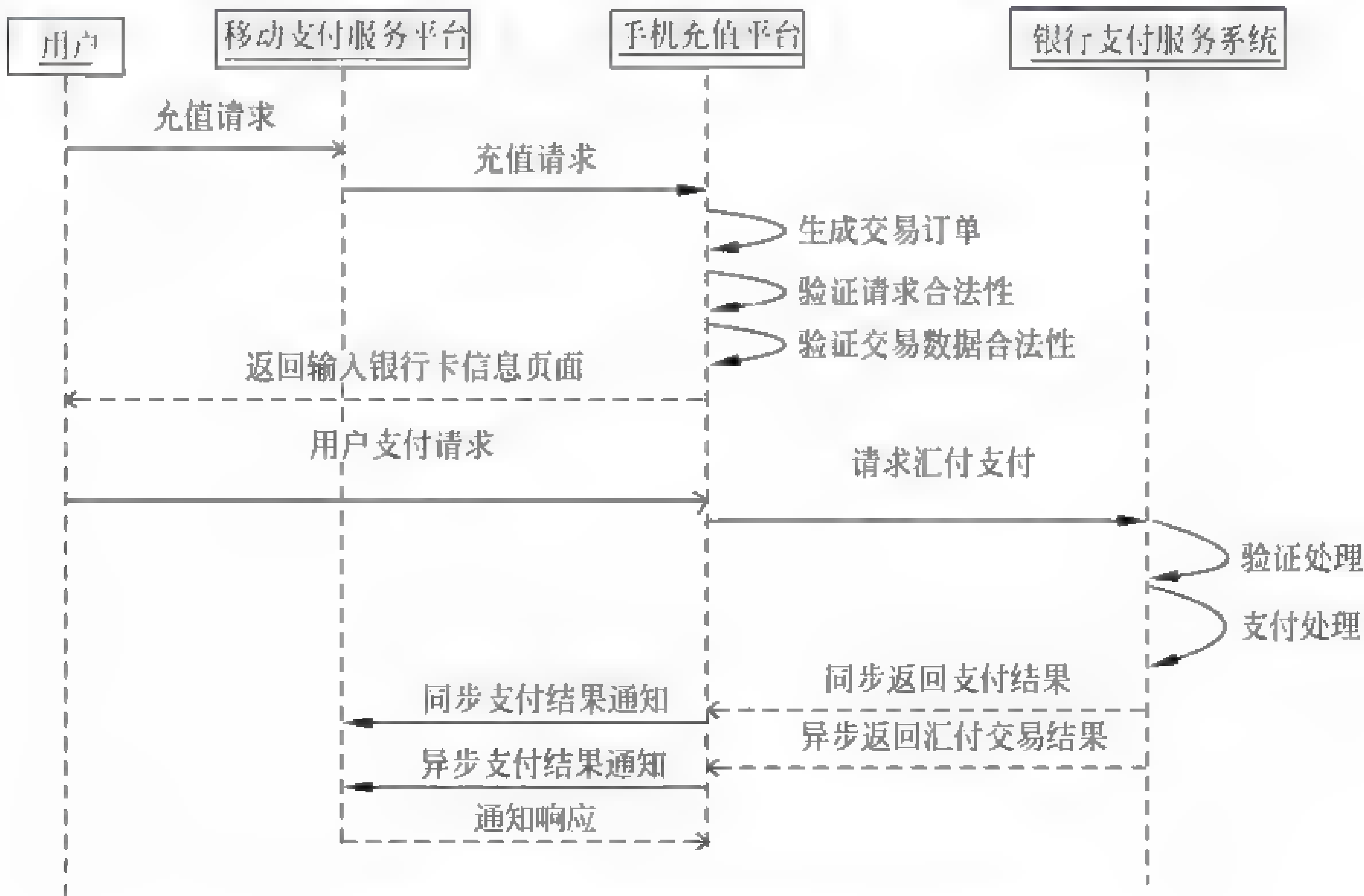

图 3-22　面向手机支付平台的时序图

（4）状态图（State Chart Diagrams）。状态图（State Diagram）是对类的描述的补充。它用于显示类的对象可能具备的所有状态，以及那些引起状态改变的事件。对象的一个事件可以是另一个对象向其发送的消息，例如到了某个指定的时刻，或者已经满足了某条件。状态的变化称为转换（Transition）。一个转换也可以有一个与相连的动作，后者用以指定完成该状态转换应该执行的操作。

状态图表示某个类所处的不同状态和该类的状态转换信息。有人可能会争论说每个类都有状态，但不是每个类都应该有一个状态图。状态图的符号集包括五个基本元素：初始起点，它使用实心圆来绘制；状态之间的转换，它使用具有开箭头的线段来绘制；状态，它使用圆角矩形来绘制；判断点，它使用空心圆来绘制；以及一个或者多个终止点，它们使用内部包含实心圆的圆来绘制。要绘制状态图，首先绘制起点和一条指向该类的初始状态的转换线段。状态本身可以在图上的任意位置绘制，然后只须使用状态转换线条将它们连接起来。

（5）活动图（Activity Diagram）。活动图（Activity Diagram）用于显示一系列顺序的活动。尽管活动图也可以用于描述像用例或交互这类的活动流程，但是一般来说，它主要还是用于描述在一个操作内执行的那些活动。活动图由多个动作状态组成，后者包含将被执行的活动（即一个动作）的规格说明。当动作完成后，动作状态将会改变，转换为一个新的状态（在状态图内，状态在进行转换之前需要标明显式的事件）。于是，控制就在这些互相连接的动作状态之间流动。同时，在活动图中也可以显示决策和条件，以及动作状态的并发执行。另外，活动图也可以包含那些被发送或接收的消息的规格说明，这些消息是被执行动作的一部分。图 3-23 是一个新生报到的活动图。

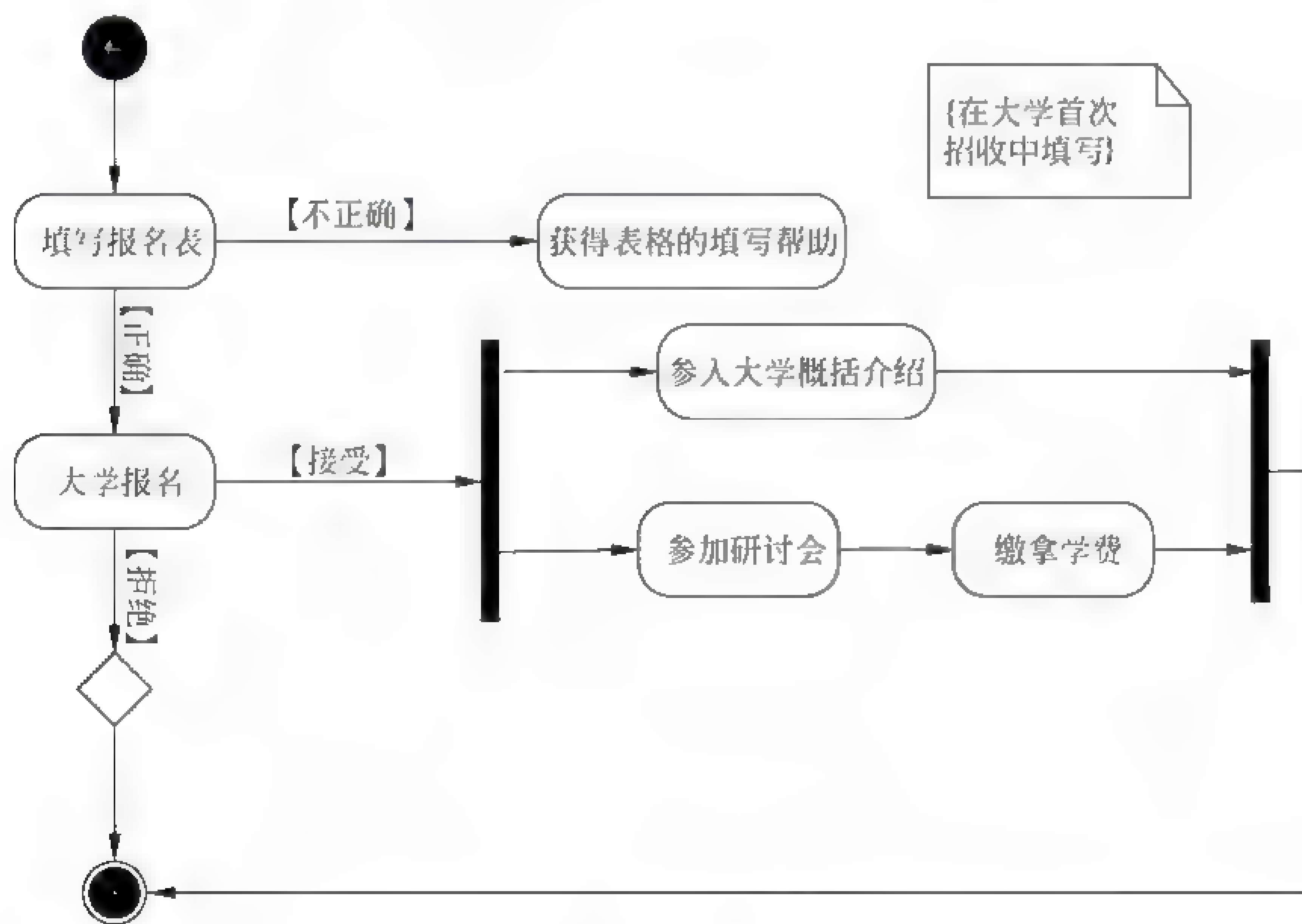

图 3-23　活动图

像状态图一样，活动图也从一个连接到初始活动的实心圆开始。活动是通过一个滑边矩形（活动的名称包含在其内）来表示的。活动可以通过转换线段连接到其他活动，或者连接到判断点，这些判断点连接到由判断点的条件所保护的不同活动。结束过程的活动连接到一

个终止点（就像在状态图中一样）。作为一种选择，活动可以分组为 swimlane，它用于表示实际执行活动的对象。

（6）元件图（Component Diagram）。又称组件图，是用代码组件来显示代码物理结构的。其中，组件可以是源代码组件、二进制组件或一个可执行的组件。因为一个组件包含它所实现的一个或多个逻辑类的相关信息，于是就创建了一个从逻辑视图到组件视图的映射。根据组件图中显示的那些组件之间的依赖关系，可以很容易地分析出其中某个组件的变化将会对其他组件产生什么样的影响。另外，组件也可以用它们输出的任意的接口来表示，并且它们可以被聚集在包内。一个组件封装了行为，实现了特定接口，如图 3-24 所示。

图 3-24　组件图结构

在以组件为基础的开发（CBD）中，组件图为架构师提供一个开始为解决方案建模的自然形式。组件图允许一个架构师验证系统的必需功能是由组件实现的，这样确保了最终系统将会被接受。同时，组件图对于不同的软件研发小组是有用的交流工具，图可以呈现给关键项目发起人及实现人员。通常，当组件图将系统的实现人员连接起来的时候，组件图通常可以使项目发起人感到轻松，因为图展示了对将要被建立的整个系统的早期理解。这样，开发者发现组件图是有用的，因为组件图给他们提供了将要建立的系统的高层次的架构视图，这将帮助开发者开始建立实现的路标，并决定关于任务分配及（或）增进需求技能。系统管理员发现组件图是有用的，因为他们可以获得将运行于他们系统上的逻辑软件组件的早期视图。

（7）协作图（Collaboration Diagram）。协作图像序列图一样显示动态协作。为了显示一个协作，通常需要在序列图和协作图间做选择。除了显示消息的交换（称为交互）以外，协作图也显示对象以及它们之间的关系（上下文）。通常，选择序列图还是协作图的决定条件是：如果时间或顺序是需要重点强调的方面，那么选择序列图；如果上下文是需要重点强调的方面，那么选择协作图。序列图和协作图都用于显示对象之间的交互。即协作图也是一种交互图，不过它主要强调了收发消息的对象的结构组织的交互图。

首先，序列图和协作图都是交互图，而且序列图和协作图是同构的，这也就意味着它们是可以相互转换的。另外一点需要注意的事，在系统建模的时候应该使交互图专注于整个系统的动态视图，这是因为 UML 中的交互图本质上展现了一种交互，而这种交互是由一组对象和它们之间的关系组成的，包括了在它们之间可能发送的消息。

（8）对象图（Object Diagram）。展现了一组对象以及它们之间的关系。对象图描述了在类图中所建立的事物的实例的静态快照。虽然对象图也给出了系统的静态设计视图或者静态进程视图，但它们都是从真实的或原型案例的角度建立起来的。

对象图是类图的一个变体，它使用的符号与类图几乎一样。对象图和类图之间的区别是：对象图用于显示类的多个对象实例，而不是实际的类。所以，对象图就是类图的一个实例，显示系统执行时的一个可能的快照——在某一时间点上系统可能呈现的样子。

（9）部署图（Deployment Diagram）。展现了对运行时处理结点以及其中的构件的配置。通过构造系统的部署图，就可以构造出整个系统体系结构的静态实施视图。即用于显示系统中的硬件和软件的物理结构。这些部署图可以显示实际的计算机和设备（结点），同时还有它们之间的必要连接，也可以显示这些连接的类型。另外，部署图也可以显示组件之间的依赖关系。

3.2.3.4　UML 的应用举例

前面详细描述了 UML，下面以一个 UML 与 XML 的转化的例子进一步描述 UML 的应用方法。

XML 模式提供了一组对 XML 文档的词汇表和语法进行约束和形式化的功能强大的工具或语言。同样可以采用面向对象的模式来设计 XML 文档，基本满足面向对象中所满足的属性，因此采用 UML 来设计 XML 提供了直接依据。下面以一个常见的网上购书系统为例进行说明 UML 设计 XML 的方法，图 3-25 是网上购书的 UML 图[①]。

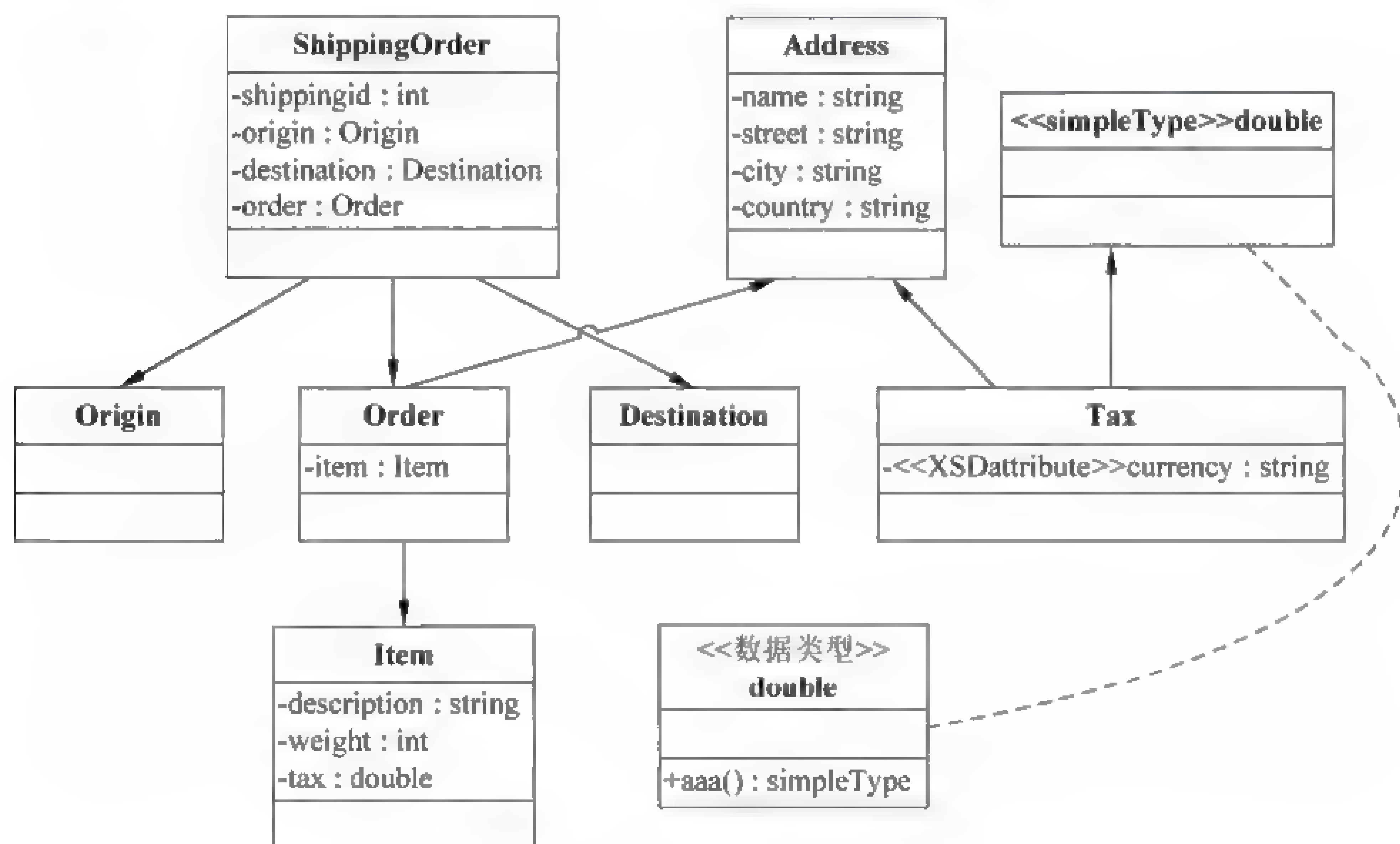

图 3-25　网上购书的 UML 图

在图 3-25 中，UML 图里描述了 Shipping Order 的业务定义，并且 Shipping Order 定义为包含 ShippingId、Origin、Destination 和 Order。此外，UML 图也用来表示组成 Origin 或 Order 的内容，并且显示的 Origin 和 Destination 的类型与类型 Address 相同，将 Address：Name、Street、City 和 Country 存储在其数据库中，这些都是业务概念，这样就让数据库模型、软件程序以及供经理和业务伙伴们访问这些属性和内容。同时，这些概念还包括基数（Order 可以包含许多 Item）、继承（Origin 继承 Address 的全部特征）以及依赖关系（Order 依赖于其 Item 的详细信息），但 UML 图能够捕获了所有这些关系。下面通过 ShippingOrder 为例来描述 UML 设计 XML 的方法，其中必须要使用到 XMI（Xml Metadata Interchange），而 XMI

① http://www.ibm.com/developerworks/cn/xml/x-umlschem/index.html

是由与供应商无关的对象管理组织（Object Management Group，OMG）赞助的，且结合了 XML、UML、MOF（Meta Object Facility）三个标准，因而代表了将元数据从一个库传送到另一个库的一种新方法；这时就可以方便使用 ArgoUML 创建 XMI 文件。如下程序清单就是 UML 与 XMI 的转换模式：

程序清单 3-1　UML sketch converted to XMI：

```
<?xml version="1.0" encoding="UTF-8" ?>
        <!DOCTYPE XMI SYSTEM "UMLX13.dtd" [      ...       ]>
  <XMI xmi.version="1.0">
     <XMI.header>
        <XMI.documentation>
           <XMI.exporter>Ideogramic UML</XMI.exporter>
              <XMI.exporterVersion>2.0 beta1</XMI.exporterVersion>
        </XMI.documentation>
          <XMI.metamodel xmi.name="UML" xmi.version="1.3"/>
      </XMI.header>
        <XMI.content>
         <Model Management.Model xmi.id="id1000133259 868model0">
<Foundation.Core.ModelElement.name>Untitled</Foundation.Core.ModelElement
.name>
            ...
                 <Foundation.Core.Namespace.ownedElement>
                    <Foundation.Core.Class xmi.id="id1000133264 1078class0">
<Foundation.Core.ModelElement.name>Small</Foundation.Core.ModelElement.name>
                ...
                    </Foundation.Core.Class>
                 </Foundation.Core.Namespace.ownedElement>
              </Model Management.Model>
              <Diagramming.Diagram xmi.id="id1000133259 877classDiagram0">
<Diagramming.Diagram.name>Main</Diagramming.Diagram.name>
                 <Diagramming.Diagram.toolName>Rational Rose
98</Diagramming.Diagram.toolName>
<Diagramming.Diagram.diagramType>ClassDiagram</Diagramming.Diagram.diagra
mType>
<Diagramming.Diagram.style>0.15,7333.33333333,8000.0,20000
20000,</Diagramming.Diagram.style>
                 <Diagramming.Diagram.owner>
                    <Foundation.Core.ModelElement
xmi.idref="id1000133259 868model0"/>
                 </Diagramming.Diagram.owner>
                 <Diagramming.Diagram.element>
                    <Diagramming.DiagramElement
xmi.id "id1000133264 1080classView0">
<Diagramming.DiagramElement.geometry>3360.0,3496.0,416.0,249.6,</Diagramm
ing.DiagramElement.geometry>
```

```
                ...
                        <Foundation.Core.PresentationElement.subject>
                           <Foundation.Core.ModelElement
xmi.idref-"id1000133264 1078class0"/>
                        </Foundation.Core.PresentationElement.subject>
                     </Diagramming.DiagramElement>
                  </Diagramming.Diagram.element>
               </Diagramming.Diagram>
            </XMI.content>
          ...
        </XMI>
```

下面的程序清单 3-2 是 ShippingOrder.xsd。程序清单 3-3 是 ShippingOrder.xml。程序清单 3-4 是数据构造 XSD。程序清单 3-5 是由 hyperModel 为图 3-25 的 UML 模型（使用 ArgoUML 创建的）所生成的部分 XMI。其中，hyperModel 是专门用于从 UML 生成 XML 的图形化工具，可以与 UML1.3 一起使用，而且能与 ArgoUML 很好的结合，即 ArgoUML 创建了一个包含 XMI 文件的项目文件，当将它导入 hyperModel 中时，就能生成了适当的模式构造。程序清单 3-6 是 ArgoUML 生成并被放入 hyperModel 的 XMI 返回了的 XML 模式构造。

程序清单 3-2　ShippingOrder.xsd：

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified">
  <xs:include schemaLocation="DataTypes2.xsd"/>
  <xs:element name="shippingOrder">
    <xs:complexType>
   <xs:sequence>
      <xs:element name="shippingId"/>
      <xs:element name="origin" type="Origin"/>
      <xs:element name="destination" type="Destination"/>
      <xs:element name="order" type="Order"/>
   </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

程序清单 3-3　ShippingOrder.xml：

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified">
  <xs:complexType name="Order">
     <xs:sequence>
```

```
            <xs:element name "item" type "Item" maxOccurs "unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name-"Item">
        <xs:sequence>
            <xs:element name="description" type="xs:string"/>
            <xs:element name="weight" type="Weight"/>
            <xs:element name="tax" type="Tax"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="Address" abstract="true">
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="street" type="xs:string"/>
            <xs:element name="city" type="xs:string"/>
            <xs:element name="country" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="Origin">
        <xs:complexContent>
            <xs:extension base="Address"/>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="Destination">
        <xs:complexContent>
            <xs:extension base="Address"/>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="Tax">
        <xs:simpleContent>
            <xs:extension base="xs:double">
                <xs:attribute name="currency" type="xs:string" use="required"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
    <xs:complexType name="Weight">
            <xs:simpleContent>
                <xs:extension base="xs:double">
                    <xs:attribute name="unit" type="xs:double" use="required"/>
                </xs:extension>
            </xs:simpleContent>
    </xs:complexType>
</xs:schema>
```

程序清单 3-4　数据构造 XSD:

```
<?xml version "1.0" encoding "UTF 8"?>
<shippingOrder xmlns:xsi "http://www.w3.org/2001/XMLSchema instance"
```

```
xsi:noNamespaceSchemaLocation "C:\\schemas\\ShippingOrder.xsd">
  <shippingId>09887</shippingId>
  <origin>
     <name>Ayesha Malik</name>
     <street>100 Wall Street</street>
     <city>New York</city>
     <country>USA</country>
  </origin>
  <destination>
     <name>Mai Madar</name>
     <street>Liivalaia 33</street>
     <city>Tallinn</city>
     <country>Estonia</country>
  </destination>
  <order>
     <item>
        <description>Ten Strawberry Jam bottles</description>
        <weight unit="kg">3.14</weight>
        <tax currency="US">7.16</tax>
     </item>
  </order>
</shippingOrder>
```

程序清单 3-5　由 hyperModel 为图 3-25 的 UML 模型（使用 ArgoUML 创建的）所生成的部分 XMI：

```
<Foundation.Core.ModelElement.name>ShippingOrder</Foundation.Core.ModelEl
ement.name>
   <Foundation.Core.ModelElement.isSpecification xmi.value="false"/>
   <Foundation.Core.GeneralizableElement.isRoot xmi.value="false"/>
   <Foundation.Core.GeneralizableElement.isLeaf xmi.value="false"/>
   <Foundation.Core.GeneralizableElement.isAbstract xmi.value="false"/>
```

程序清单 3-6　Argo UML 生成并被放入 hyperModel 的 XMI 返回了的 XMI 模式构造：

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified">
   <!-- Class: ShippingOrderType  -->
  <xs:element name="ShippingOrder" type="ShippingOrderType"/>
  <xs:complexType name="ShippingOrderType">
     <xs:sequence>
        <xs:element name="shippingId" type="xs:int"/>
        <xs:element name="origin">
           <xs:complexType>
              <xs:sequence>
                 <xs:element ref="Origin"/>
```

```
                </xs:sequence>
              </xs:complexType>
            </xs:element>
            <xs:element name="destination">
              <xs:complexType>
                <xs:sequence>
                  <xs:element ref="Destination"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
            <xs:element name="order">
              <xs:complexType>
                <xs:sequence>
                  <xs:element ref="Order"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:schema>
```

3.3 面向服务计算的软件分析设计方法

面向服务的方法是目前正在以快速的方式应用到各领域，为解决分布式应用的互操作性和异构性带来了前所未有的改变和高可伸缩性。它还是一种面向服务的架构的技术，通过标准的 Web 协议提供服务，目的是保证不同平台的应用服务可以互操作。并以服务为基础形成了一种新型的软件分析设计方法，并将业务逻辑发布成一个安全的服务来实现不同的需求请求，而软件本身则根据用户的需求，就是将业务逻辑以服务的形式进行研发，从而使整个软件系统不再是一个一个难以扩展的业务，难以实现松散耦合的内部程序结构。同样，作为面向开源软件的软件的方法，也与服务相关，就从软件开发技术角度来讲，就有多款支持面向服务开发的开源软件，如 Axis、CXF。

3.3.1 面向服务的分析设计方法概述

近几年来，因特网从最初的简单的信息存储发展到今天的错综复杂的各种电子商务、电子政务等方面的应用。这些错综复杂的应用会涉及到不同系统、不同平台、不同应用和不同部门的信息，甚至涉及到不同企业间的异构分布式交易。因而如何集成不同的异构的分布式的组件、系统、应用和网络成为了一个非常棘手但又必须解决的难题。幸运地是，基于 XML 技术的 Web 服务的出现为解决这一问题提供了最佳手段和契机。Web 服务是一种通用开放标准、Internet 及基于业界标准的 Intranet 技术动态交互的应用程序，它可以将不同厂商、不同

硬件、不同语言编写成的应用程序集成到一起。Web 服务建立在现有的和新兴的标准之上，例如，超文本传输协议（HyperText Transfer Protocol，HTTP）、可扩展置标语言（Extensible Markup Language，XML）、简单对象访问协议（Simple Object Access Protocol，SOAP）、Web 服务描述语言（Web Service Description Language，WSDL）以及通用描述、发现和集成（Universal Description Discovery and Integration，UDDI），图 3-26 所示是 Web 服务基础结构，并且 Web 服务是由许多规范来支持其标准化。同时，Web 服务系列标准是一组新兴标准，支持异类信息技术流程和系统间的互操作集成，如图 3-27 所示是 Web 服务支持的主要标准结构。可以将其视为一种新的具有自包含性和自描述性的 Web 应用程序，能提供从最基本的到最复杂的业务和科学流程的功能和互操作机制。简而言之，Web 服务系列标准承诺提供用于在异类系统间进行互操作集成的公共标准机制，实际上，其关键之处在于标准化。

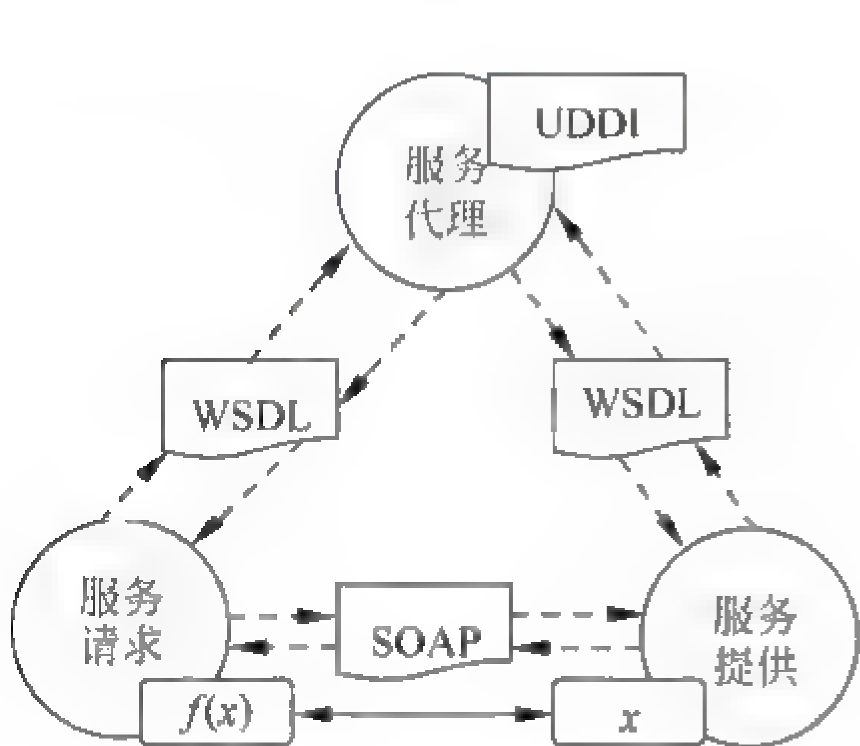

图 3-26　Web 服务基本结构

图 3-27　Web 服务规范结构

而根据 W3C 的定义，Web 服务（Web Service）应当是一个软件系统，用以支持网络间不同机器的互动操作。网络服务通常是许多应用程序接口（API）所组成的，它们通 Internet 的远程服务器端，执行客户所提交服务的请求。尽管 W3C 的定义涵盖诸多相异且无法介分的系统，不过通常指有关主从式架构（Client-server）之间根据 SOAP 协议进行传递 XML 格式消息。无论定义还是实现，Web 服务过程中会由服务器提供一个机器可读的描述（通常基于 WSDL）以辨识服务器所提供的 Web 服务。另外，虽然 WSDL 不是 SOAP 服务端点的必要条件，但目前基于 Java 的主流 Web 服务开发框架往往需要 WSDL 实现客户端的源代码生成。一些工业标准化组织，比如 WS-I，就在 Web 服务定义中强制包含 SOAP 和 WSDL。

在 Web 服务出现之前，在大多数系统上，采用的是固定的接口，但对于环境或需要的改变，这缺乏灵活性或适用性。Web 服务所使用的 XML 可以用真正与平台无关的方式来描述任何（所有）数据，以跨系统交换数据，因此转向了松耦合应用程序。而且，Web 服务可以在较抽象的层面上工作，较抽象层面可以按照需要动态地重新评估、修改或处理数据类型。所以，从技术层面上讲，Web 服务可以更方便地处理数据，并且允许软件更自由地进行通信。而 Web 服务也是一个软件接口，它描述了一组可以在网络上通过标准化的 XML 消息传递访问的操作。它使用基于 XML 语言的协议来描述要执行的操作或者要与另一个 Web 服务交换的数据。一组以这种方式交互的 Web 服务在 SOA 中定义了特殊的 Web 服务应用程序。Web

服务采用一系列的相关协议来描述、传递服务和与服务交互。SOAP 协议对消息进行编码，这样就可以通过传输协议（如 HTTP、IIOP、SMTP 或其他协议）在网络上传递它们。WSDL 表示为一系列 XML 语句，这些语句组成了每个服务的接口的定义。UDDI 定义了服务如何公开它们自己以及如何在网络上相互发现。并且 Web 服务是独立的模块化的应用程序，它们常常不能利用自身的力量满足业务流程的操作需求。为满足日益复杂多变的业务需求，需要将这些 Web 服务链接在一起成为一个业务流程来实现更复杂的功能。业务流程指定了一组 Web 服务的操作的可能执行顺序以及这些 Web 服务间共享的数据等。

下面的内容就是以服务和 UML 建模为基础概述 SOA、BPEL、ESB 和 SoaML，相关的 Web 服务技术详细的介绍见下面的章节。

3.3.2 面向服务体系结构的设计方法

面向服务的体系结构（Service-Oriented Architecture，SOA）是一种用于创建企业 IT 体系结构的体系结构样式，利用了面向服务的原则来实现业务与支持业务的信息系统之间的更为紧密的关系。下面就从 SOA 基础、SOA 场景、SOA 成熟度、SOA 设计原则和 UML 与 SOA 的关系五个角度展开论述 SOA 的软件设计方法。

3.3.2.1 SOA 基础

SOA 支持将业务转换为一组相互链接的服务或可重复业务任务，可在需要时通过网络访问这些服务和任务。这个网络可以是本地网络、Internet，也可以分散于各地且采用不同的技术来通过对世界各地的服务进行组合，最终可让用户感觉似乎这些服务就像安装在本地桌面上一样。同时，可以对这些服务进行结合，以完成特定的业务任务，从而让业务快速适应不断变化的客观条件和需求。当在战略业务目标的引导下进行 SOA 实现时，可确保对业务进行积极转换，具有 IT 与业务的一致性和 IT 资产的最大化重用的好处。并且有助于确保在耗资巨大的 IT 项目中的投资能够给业务带来长远的价值。然而，怎样确保每个基于 SOA 的解决方案都能提供真正的业务价值呢。这时，诸如微软、IBM 等软件巨头就提出了 SOA 的创造价值的方法，下面就以 IBM 的模式来进一步说明：

IBM 定义的五个切入点（均基于实际的客户经验确定）帮助业务实现预定义的 SOA 解决方案，从而从中获益。这些切入点同时受到业务需求（人员、流程和信息切入点）和 IT 需求（连接性和重用切入点）的驱动。

（1）人员。SOA 的这个切入点关注用户体验，以帮助生成、调用和实现更好的协作，从而获得一致的人员与流程交互，提高业务效率。例如，通过使用 SOA，可以创建基于服务的 Portlet 来提高此协作。

（2）流程。流程切入点可帮助企业了解其业务中发生的情况，从而支持其对现有业务模型进行改进。通过使用 SOA，可以将业务流程转换为可重用且具有灵活性的服务，从而改进和优化这些新流程。

（3）信息。通过使用 SOA 的这个切入点，能以一致而可见的方式利用公司中的信息。通过在所有业务领域提供这个一致而受信任的信息，也可促进企业各个领域的创新工作，从而更为有效地进行竞争。并且通过使用 SOA，可以更好地控制信息，而且通过信息与业务流程的结合，可以发现很多有意义的新关系。

（4）连接性。利用连接性这个切入点，可以有效地连接基础设施，从而将企业中的所有人员、流程和信息整合到一起。通过在服务间和整个环境中实现灵活的 SOA 连接，也可以获取现有业务流程并在不需要太多工作的情况下，通过其他业务通道提供此流程，甚至还能以安全的方式连接防火墙外的外部合作伙伴。

（5）重用。通过 SOA 重用服务，可以充分利用企业中已经存在的服务。通过对现有资源进行构建；也可以简化业务流程，在整个企业内确保一致性并缩短开发时间。所有这些将能帮助节约大量的时间和资金。

3.3.2.2　SOA 场景

以定义切入点的原因是为了帮助客户了解如何认识 SOA。不过，还需要进一步的实现细节来帮助客户的业务和 IT 团队来创造 SOA 的价值。而这正是需要更为具体的场景的原因，并且每个场景都提供了经过测试和集成的产品或实现，用于实现此场景。因此，可以将这些场景映射到具体的目标和需求，从而很好地确定自己如何实现这些好处。

1．服务创建

创建灵活的、基于服务的业务应用程序。新的面向服务的应用程序将业务行为作为服务公开，同时还能重用且作为服务公开的业务逻辑。

2．服务连接性

无论何时何地使用何种工具，都能使用中间层服务网关或总线让各种应用程序访问核心服务集，从而通过无缝的消息和信息流将企业中的人员、流程和信息连接起来。

3．交互与协作服务

必须通过多种设备（如浏览器、PC 和移动设备）向用户提供一个或一组服务。交互与协作服务还可通过将这些服务聚合为视图，以交付信息并在业务流程的上下文进行交互，从而提高人员工作效率。

4．SOA 所支持的业务流程管理

业务流程管理是将软件功能和业务专业知识相结合来加速流程改进和促进业务创新的学科。

5．作为服务的信息

“作为服务的信息”可在企业内作为可重用服务访问复杂的异类数据源。

6．SOA 设计

通过一组角色、方法和构件保持业务设计建模和 IT 解决方案设计的一致，以提供一组供优化的显式业务流程和用于组合及集成的服务。

7．SOA 治理

建立并执行 SOA 开发与运行时流程。定义策略、流程和工具来监视服务的归属，使用人和使用方式来缩短软件研发时间。

8．SOA 安全性和管理

它作为 IT 服务管理（IT Service Management，ITSM）服务一部分的发现、监视、保护、

供应、更改和生命周期管理工作。

3.3.2.3 SOA成熟度探讨

软件工程协会（Software Engineering Institute）于1991年引入了CMM的1.0版。CMM是一个用于描述软件流程成熟度的原则和实践的模型，作为评估软件流程成熟度的基准得到了广泛的认可。该模型的目标是使得软件流程具有更高的可预测性和可重复性，从而提高信息技术（IT）组织提供软件产品或项目的效率。这样，软件开发的流程成熟度模型（如CMM）可以帮助确定组织内的SOA需求，还能够帮助确定SOA的成本和好处，从而为项目带来稳定的回报。

CMM定义了一个模型，各种组织可以使用此模型来评估其软件流程成熟度，它还定义了一个可以用于从一个级别上升到另一个级别的模型。CMM描述的五个成熟度级别可以由每个级别所做的主要流程更改加以描述。表3-6所示概述了SOA的成熟度的特征和影响。

表3-6 SOA成熟度级别

级别	特征	影响
第1级：初始化	没有正式软件开发流程。 只存在很少的体系结构文档。 项目团队之间不进行通信	项目之间不具有体系结构一致性。 难于理解和修改生成的系统。 只存在很少的可重用构件。 团队为每个项目都重复相同的工作
第2级：可重复	有一些体系结构文档。 体系结构在项目团队内执行。 项目团队之间有临时的体系结构通信	相对于第1级而言，有一些小改进 一些成功的实践是可重复的。 认识到EA工作可能很有价值
第3级：已定义	配备了EA团队，该团队定义了参考体系结构和一些软件开发实践。 鼓励项目团队使用此结构，但不会因为使用此结构而得到奖励。 EA并不满足每个业务范围（LOB）的所有需求	难于达到一致，即EA团队和项目团队的协作不甚理想。 体系结构维护的问题很大。 通常体系结构的有效期为6～12个月
第4级：已管理	SOA被认为是体系结构活动的终结点。 业务范围（LOB）团队定义了一个SOA。 配备了支持和控制模型。 LOB会因公开和使用服务而得到奖励	早期的成本似乎太高昂。 它降低了由于体系结构层不一致而导致项目延迟的风险。 组织内的SOA看起来好像有一些冲力
第5级：优化中	SOA成为一个起点。 组织希望探索与其客户、供应商和合作伙伴相关的服务定向。 有持续的体系结构优化	业务具有灵活性。 能与来自客户、合作伙伴、供应商和其他方面的服务进行互操作。 推向市场的时间更快。 总体拥有成本（TCO）更低

第1级　初始化

指组织通常没有正式的体系结构流程，即体系结构没有从项目分离出来。通常，这些组织不具有EA（Enterprise Architecture）团队；每个项目团队通常根据业务范围（LOB，Line of business）进行划分，并彼此独立地进行工作。精力主要放在交付单个项目上。

此级别的结果包括项目计划不可预测、预算超支，而且代码质量差（通常不能重用，且难于维护）。各个项目重复的相同任务不可重用，这将导致交付和维护成本的增加。

第 2 级　可重复

在此级别，进行了一些体系结构方面的工作。项目团队通常定义一个可重用体系结构在多个项目间使用。同时，项目团队之间也建立了非正式的通信渠道。当然，一个 EA 团队将帮助在较为混乱的环境中形成结构，促进项目团队间的通信；不过，在此阶段，仍然很少存在此类团队，通信是临时性的，较为混乱。

此级别的结果包括对体系结构组件的一些重用。临时流程和较为混乱的通信路线使体系结构解决方案中具有一定的可重复性，因而降低了软件的交付成本和维护成本。不过，从资金的角度而言，此成本节约不甚明显。同时，此级别的成熟度的最大优势在于实现了结构化的流程提供的优势。例如，项目团队正逐渐认识到软件开发的协同方法的潜在优势。他们认识到可以防止巨额的成本超支和创建可预测的软件开发计划，并能提高软件的总体质量。在项目团队之间创建这些临时的通信线路的支持者可以就此向管理层寻求支持，以向 EA 组织发展，或至少获得对更佳的体系结构活动的正式认可。

第 3 级　已定义

表示组织在 EA 活动方面进行了一些投资、配备了 EA 团队，为其指定了对体系结构元素进行标准化的任务，负责进行创建参考体系结构的活动，就此体系结构对项目团队进行培训，并定义控制和执行策略。通常，EA 团队将创建一组技术组件和框架，然后标准化各个项目团队间对这些框架的使用。

第 4 级　已管理

当 EA 团队开始定义 SOA 路线时，就达到了这一级别的成熟度。今天，每个大型组织都有一群架构师在谈论 SOA。最起码的这些架构师看起来已认识到 SOA 的价值，并在尝试形成 SOA 策略。

第 5 级　优化中

在此级，体系结构流程和策略都已制度化，对服务价值也有了清晰的认识。且配备了框架供每个团队公开和使用服务。同时，在此级别，组织可以真正地充分利用 SOA 的价值。使他们开始了解如何与其业务合作伙伴、供应商和客户交换服务。

3.3.2.4　SOA 设计原则

下面从重用、松散耦合、无状态性、粒度和可扩展性五个方面分析 SOA 的设计原则，这些原则在面向服务的解决方案设计中是十分重要的。图 3-28 是 SOA 的结构图。

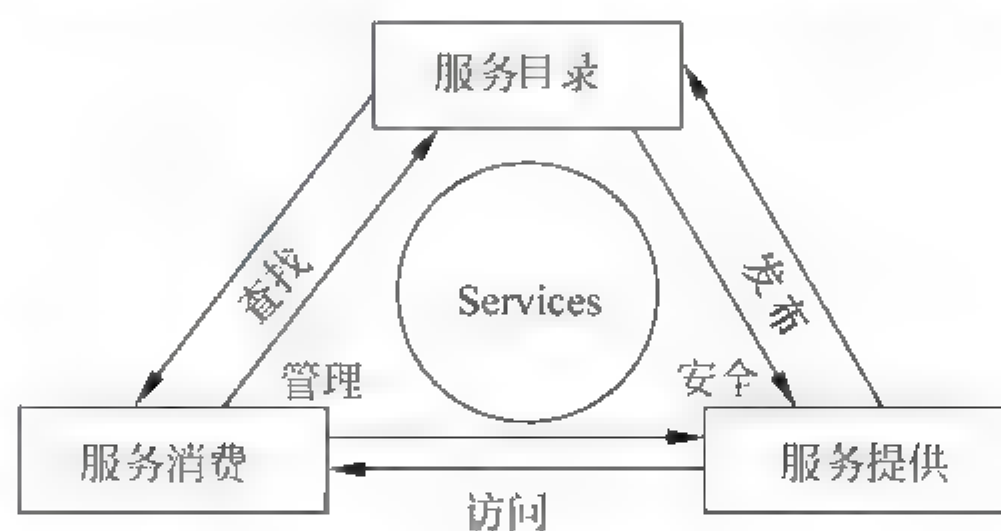

图 3-28　SOA 结构图

1．重用

在设计服务时，需要把重用这一原则随时记在心中。通常，在设计时，特定的服务使用者会有特定的要求。不过，如果想从 SOA 中获益，希望哪些需求略有不同，且其他使用者会重用自己设计的服务。而在创建设计时，并不知道这些使用者，也不知道它们有什么需求，因此这并不是一项轻松的任务。同时，服务重用是影响整个面向服务的软件系统至关重要的因素。这时可以从以下几个方面来考虑 SOA 的重用设计：

（1）根据业务域（而不是技术）为服务及其操作提供一个有意义的描述性名称。

（2）提供完整的、可由人工阅读或计算机处理的规范，并将其公布给服务注册中心，使服务可以被发现。

（3）各个使用场景的服务质量与初始场景不同，例如，如果服务被频繁使用，需求和工作负载提高，可以制订相应的计划。

（4）是否有可能对服务提供的初始功能（是根据初始需求集提供的）进行扩展，以便完善服务。

（5）是否有可能在多种不同的平台上实现某个服务规范。

2．松散耦合

耦合是指实体或系统彼此间的依存程度。在 SOA 的上下文中，可以考虑服务规范与它的提供者或实现之间的耦合，或服务提供者与服务使用者间的耦合。如要支持 SOA 应有的灵活性，必须采用松散耦合。并且服务提供者无须了解服务使用者的某个特定实例，反之亦然。如果要替换某个服务提供者或服务的实现，必须保证这样做对使用者造成的影响可忽略不计，或不会对其造成干扰。一般可以从以下几个方面加以考虑：

（1）将服务规范与服务实现分离开来。

（2）使用工具，以一种人机可读的标准格式（如 WSDL 和 WS-Policy）描述服务规范，以使代码生成器能帮助实现服务。

（3）当开发服务使用者的代码时，可能存在服务中间层或其他提供者，请不要对某个提供者的实例的任何信息进行硬编码。

3．无状态性

事务状态意味着服务必须具有关于某些发生事件的信息，这些事件是一个在服务提供者和服务使用者各自的特定实例间长期运行的事务的一部分。例如，如果服务操作已被调用，或在调用某个操作前需要先调用另一个操作，在这两种情况下调用服务操作的结果是不同的。虽然在基于组件的开发中可以找到事务状态的身影，但对于 SOA 来说，事务状态却是应该避免的。这是因为数据（信息）状态意味着需要对数据实例的状态进行管理。这是因为某些服务通常需要管理数据的状态，以保险业中的索赔服务为例，该服务必须管理索赔实例的状态，因为多个用户会通过不同的业务事务访问这些实例。

4．粒度

对于面向服务的设计，可以对粒度进行考虑，如服务提供者级或服务规范级的粒度（对于前者而言，要考虑应提供多少服务）。服务规范是一个针对服务操作的逻辑分组。在此处，逻辑表示它在业务方面的合理性。例如，在保险领域中，一个索赔处理接口将提供使用索赔处理的场景中所需的所有操作，当然，这在 IT 实现方面也是合理的。例如，服务中所有被标识出的操作都可以通过同一种开发迭代实现。

在设计服务操作时，需要考虑协作、使用场景，以及在服务提供者和使用者之间流动的消息的数目和大小。粗粒度的操作可以提供更高的业务价值，而且使对实现的修改变得更容易了，从而使得粗粒度操作的参数（使用消息作为输入、输出和故障参数）出错率较低。不过，使用细粒度的参数（而不是消息）通常具有更好的描述性。例如，如果某个操作服务签名为 DetermineEligibility（application: ApplicationMessage），则需要查看 ApplicationMessage 的定义。如果签名为 DetermineEligibility（customer :Customer, product :Product, date :Date,

amount :Amount)，则该签名的描述性更好。此外，Customer 或 Product 参数类型可以在其他操作中重用，这与 ApplicationMessage 不同。

对于服务粒度，服务是可组合的。这涉及到重用和在现有功能基础上提供新功能的能力。某一粒度级别的一组服务可以通过编排，形成另一个具有较粗粒度的服务。可以考虑这个例子：多个服务提供各种业务任务，然后将这些业务任务排成序列（一个业务流程），以提供另一个服务。

5．可扩展性

SOA 的可扩展性是提高 SOA 的重用有效的手段之一，也是提高整个 SOA 生命周期有效的方法之一。一般可以从以下几个方面加以考虑：

（1）让各种规模的组织都能使用 SOA 解决方案。

（2）更改软件部署活动到更为动态且更为省时的模型，与业务更为相配。

（3）更便于添加或更改合作伙伴。

（4）加速合并和收购。

（5）方便公开服务，而这就代表着新的收益来源。

3.3.2.5　用 UML 表示 SOA 方法

使用 UML 表示 SOA 的组件、连接以及与 SOA 体系结构模式的交互，有助于以一种逻辑格式来表示 SOA。采用一种与 UML 产品无关的方式表示 SOA 模式，在它最简单的形式中，SOA 模式由分离的企业服务总线（Enterprise Service Bus，ESB）组成，该总线可以连接请求者和提供者，并在它们之间提供交互的服务[1]，如图 3-29 所示。

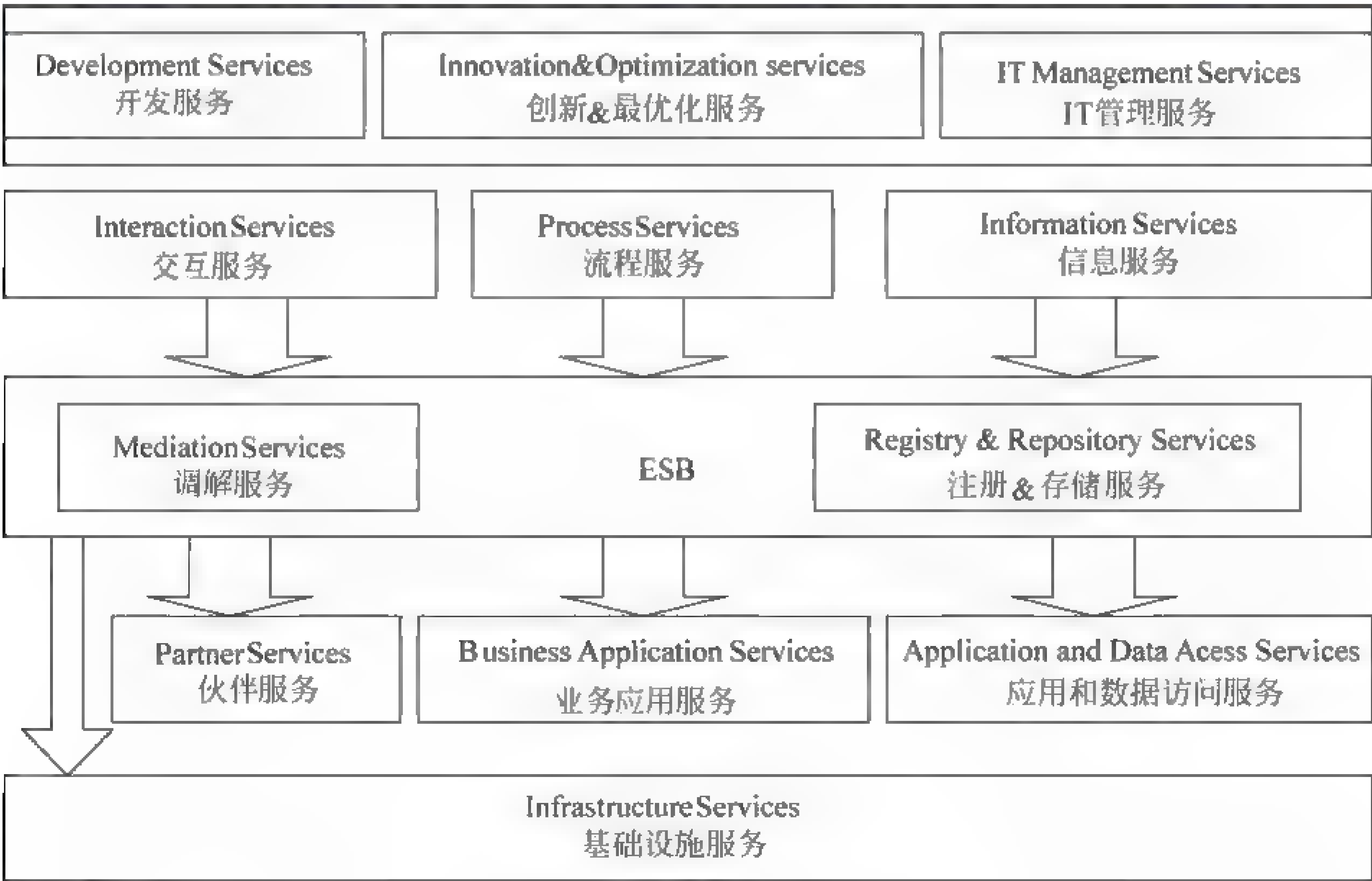

图 3-29　逻辑的 SOA 体系结构

① http://www.ibm.com/developerworks/cn/architecture/ar-logsoa

由 UML 表示 SOA 模式是由 ESB 基础结构、服务交互点（SIP）或端点组成，这里的 SIP 由交互服务、流程服务、信息服务、合作伙伴服务、业务应用程序服务和应用程序和数据访问服务构成。并实现 UML-to-SOA 的转换，而且这些转换工具包括基于 Service Component Architecture（SCA）的 SOA 转换工具，它使用 Service Component Definition Language（SCDL）来表示元数据[①]。

1．ESB

ESB 用作 SOA 模型的连接入口点，并且提供了下列服务：请求和响应服务、转换、基于内容的路由、自定义的日志记录、优化和监视。同时，ESB 还提供了各种服务的通用连接和虚拟化。为了满足最新业务应用程序的需求，ESB 充分利用了服务组件体系结构（SCA）编程模型，如图 3-30 所示。

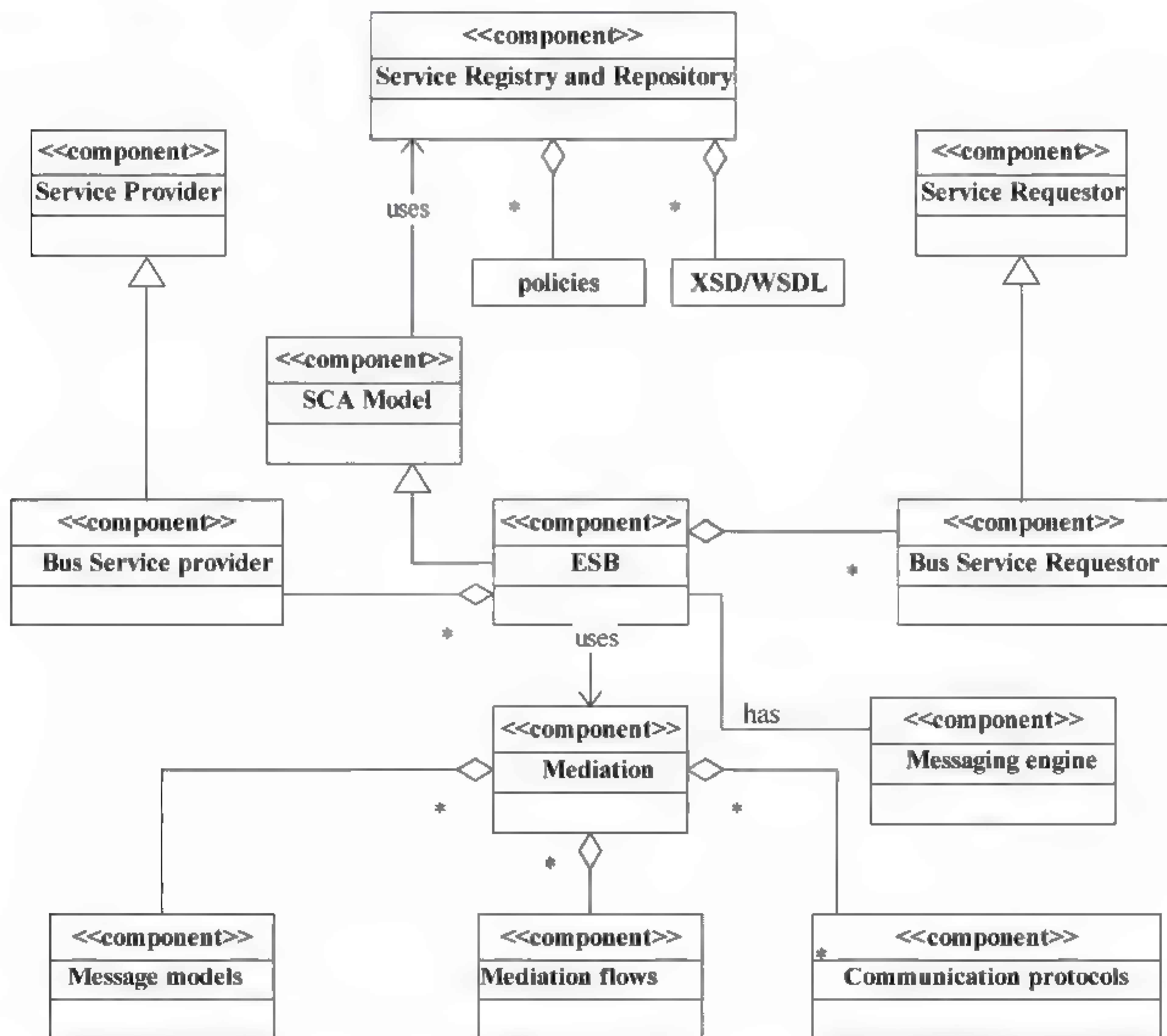

图 3-30　企业服务总线（ESB）

该 ESB 的消息交互是基于 JMS（Java Message Service）的，这个 ESB 使用了一个中介组件（模块），该组件基于 SCA 模块，以便为服务请求者和服务提供者之间的消息提供中介。从而可以定制 ESB 中的中介服务，以形成复杂的中介模式，这种中介模式采用与具体位置和标识无关的方式来实现虚拟化。它们还可以提高服务质量（QoS）需求，如性能、消息的加

① http://www.ibm.com/developerworks/cn/rational/08/0115_gorelik/

密/解密，以及可靠且安全的内容交付及事务。

2．交互服务

交互服务：是指具有 ESB 的服务集成点的交互服务，这些交互服务结点可以作为用户的 SOA 入口点，从而为 SOA 提供表示层。并对相关的接口进行了抽象，聚合了最终用户和 SOA 应用程序之间的各个信息源。其内容包括三类服务：用户接口服务（由决策制定和可视化操作的门户应用程序组成）、用户交互服务（由可视化、协作、组合应用程序、警报和表格组成）和部署服务（包含移动设备、浏览器和富客户端），如图 3-31 所示。

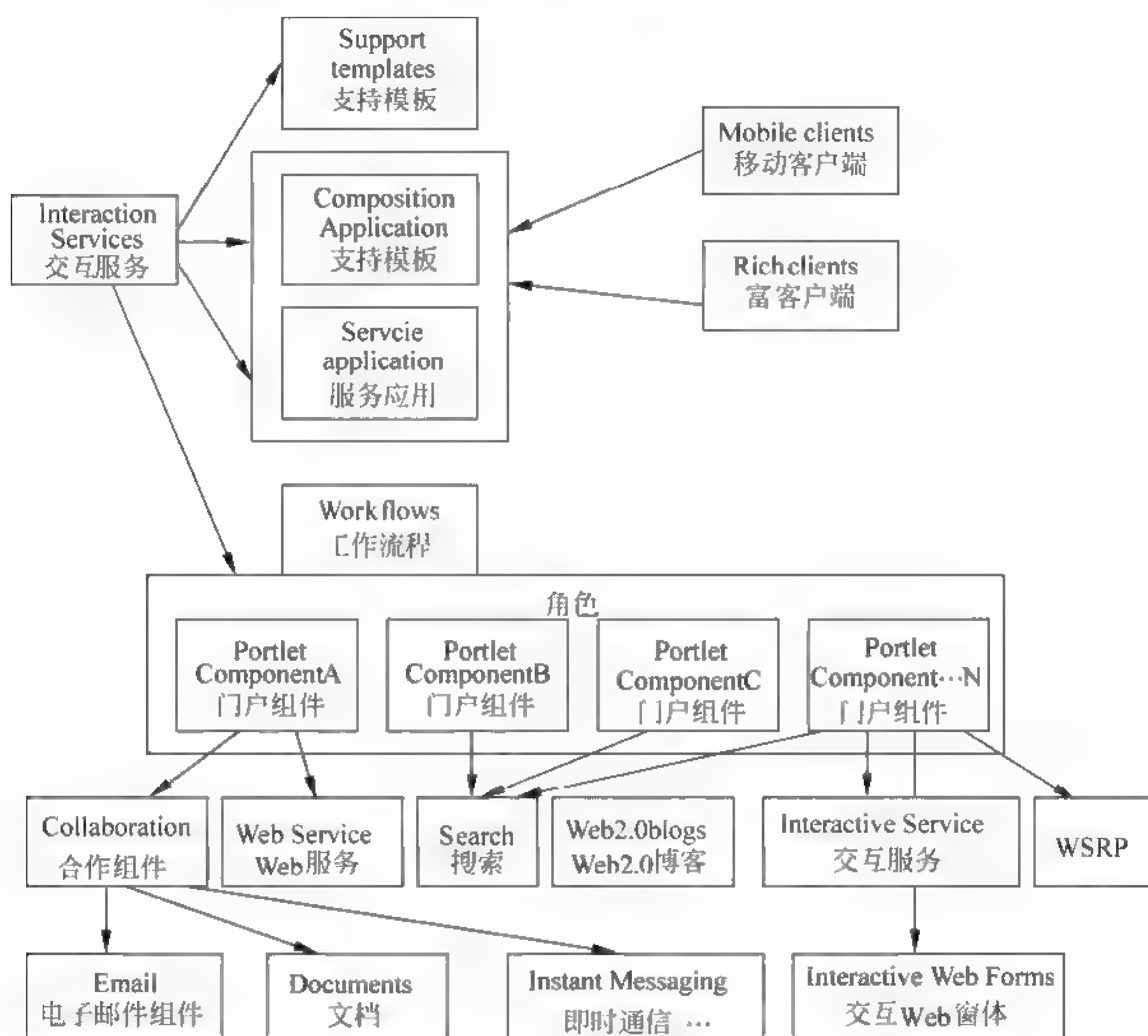

图 3-31　交互服务

交互服务使用支持的模板组件以简单地创建组合应用程序，这些组合应用程序包括：为外包的或者内部的服务应用程序提供基础；支持富客户端和移动设备最终用户客户端；提供高度自定义和动态的数据，这将提供实时的可见性，以使将结果与基础业务流程度量关联起来。且组合应用程序中的每个部分都可能包含预先构建的、具有特定功能和相关联的工作流的 Portlet。交互服务还可以具有内置的筛选功能、基于浏览器的配置向导、交互的 Web 窗体、搜索、Web 2.0 技术和协作。例如，协作服务组件是一个完全集成的、基于门户的协作环境，该环境包括电子邮件、日历和日程安排、即时消息传递、Web 会议、文档以及 Web 内容管理。

3．流程服务

流程服务是指用于在 SOA 域内执行服务功能的流程服务和组件，它使用业务流程和中

介模块来实现其他的业务流需求。并使用 SCA 编程模型对使用和产生业务数据的业务服务进行建模，如图 3-32 所示，其中菱形黑块（◆）和黑圆点（●）表示所连接两者的“复合”。

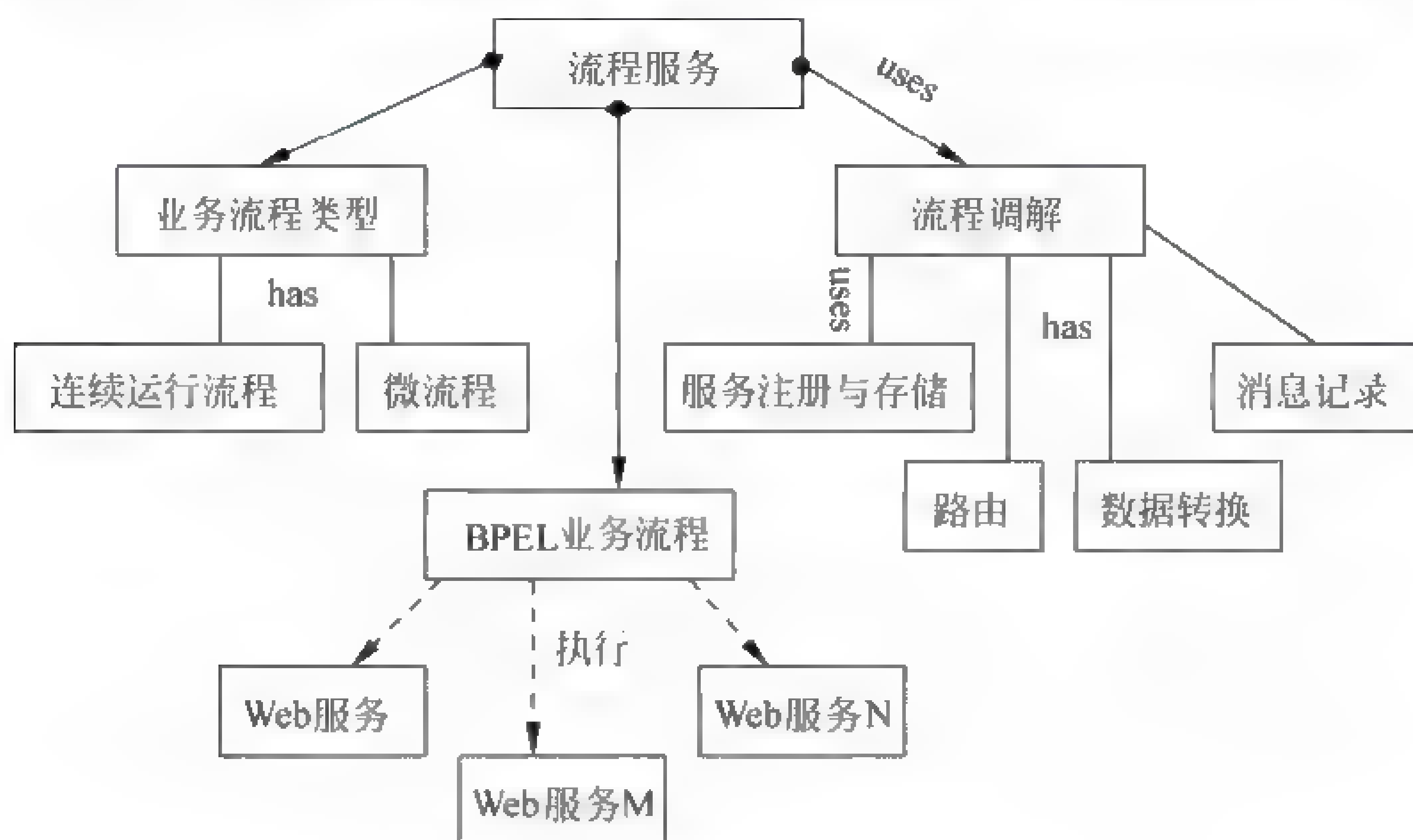

图 3-32　流程服务

流程服务可以使用业务流程执行语言（BPEL）来定义业务流程；业务流程是在预定义的序列中调用以实现业务目标的一组业务相关的活动、规则和条件；业务规则是一种通过业务功能的外部化来实现和实施业务策略的手段。而外部化支持从应用程序的其他方面独立地管理业务规则，这种独立性支持动态业务规则更新功能，从而提供了更加灵活的业务。同时，流程服务由 SCA 组件由接口、引用和实现组成。且服务组件可以包含使用 Java 编写的接口或者 WSDL 端口类型。业务流程类型组件由流程实现类型组成，这个流程实现类型可以通过 Java 接口或者 WSDL 端口类型接口实现一个或者多个 SCA 接口。流程运行的时间可能很长，也可能很短；运行时间很短的流程称为微流。运行时间很长的业务流程可以与多个合作伙伴进行交互，并且通过执行标准的、无状态的 Web 服务调用来进行交互。

在具体实现时，可以通过 Web 服务接口与各个合作伙伴发生交互，以及 BPEL 基于 WSDL 和 XML 模式构建；并且可以按 BPEL 规范定义的那样，使用一个用于语法扩展的 XML 模式，以及应用于语义约束的一组全面的规则，来完成对流程模型的验证。

4．信息服务

信息服务就是实现真正交互的各类信息。一般包括元数据管理、商业智能中的提取、转换和加载（ETL）、联合、数据布置（复制和缓存）、数据建模、搜索和分析方法，如图 3-33 所示。

（1）元数据管理。元数据（有关元数据的详细描述可参见第 6.9.2.1 小节中第 3 条）是关于数据结构和含义的信息，元数据管理组件可以管理元数据和元模型，其中元模型定义了元数据的结构和语义。标准化的元模型示例包括 UML 和公共仓库元模型（Common Warehouse Metamodel，CWM）。而元模型层由结构的描述和元数据的语义组成。它试图提供一种公共语言，以描述信息的所有其他模型。MetaObject Facility（MOF）是一种用于元模型的标准。

（2）ETL。从一个或者多个数据源提取、转换和加载数据到一个或者多个目标（ETL 支持数据整合、迁移和传播，并且它与数据仓库和业务智能功能紧密结合。）

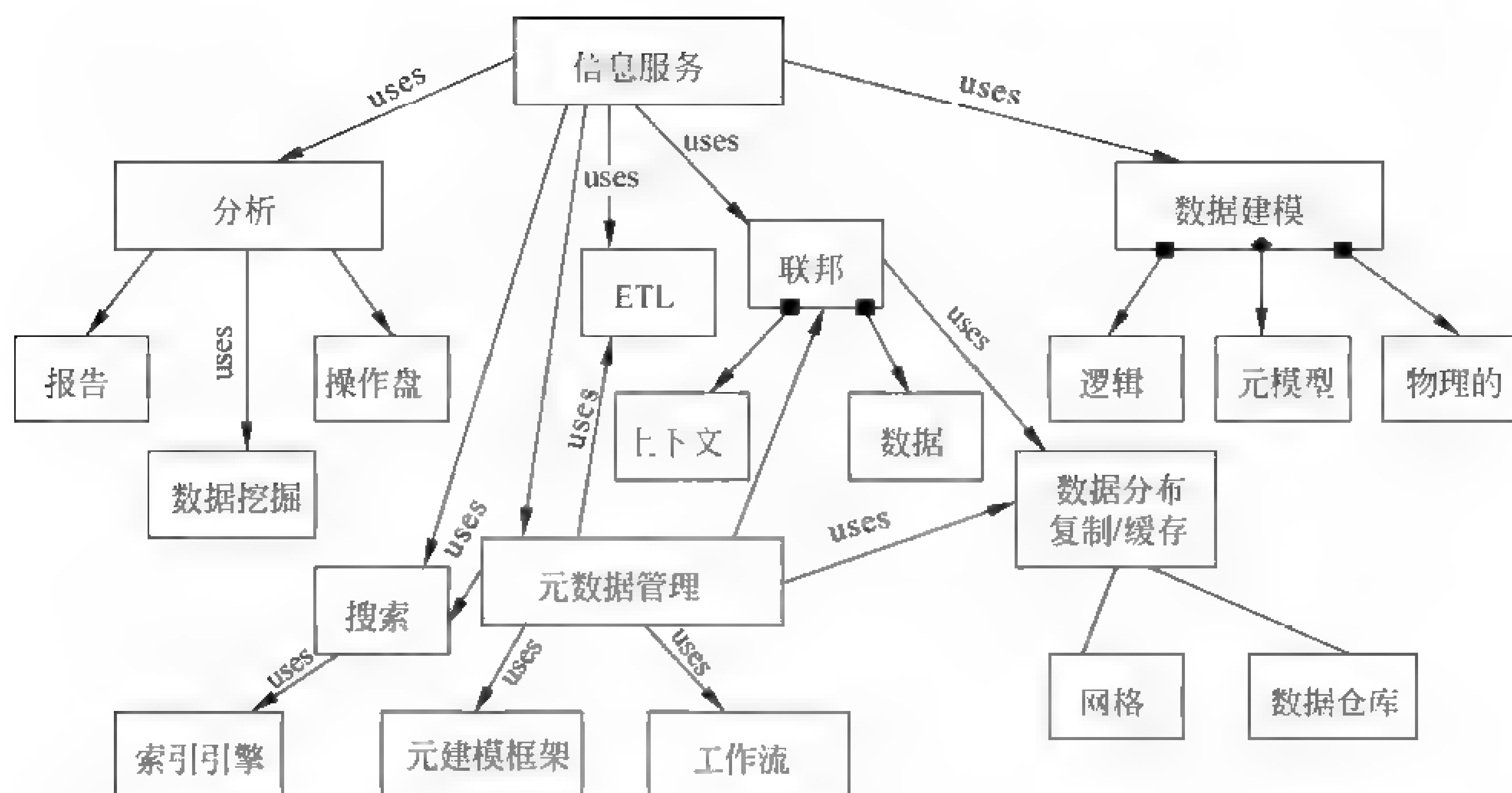

图 3-33　信息服务

（3）联合。使用数据和内容类以联合异类内容源的数据，且联合减少了为各种数据源开发和维护自定义应用程序编程接口（API）的需要。通过缓存频繁使用的数据，以及使用物化查询表（MQT）和分布式查询优化与执行，联合还可以改善性能。

（4）数据布置。从一个位置复制数据到另一个位置。

（5）数据建模。指提供了逻辑、物理和元数据模型的聚合，用于存储企业各自的模型。且对数据模型进行仔细地设计同样可以提高整体事务性能。同时，在事务类型方面存在以下依赖关系：联机事务处理（OLTP）事务使用 E/R（实体/关系）模型、数据仓库事务使用多维建模技术。

（6）搜索。最通用的搜索功能是通过一种查询语言，如 SQL 和 XQuery。数据库搜索对于检索结构化的和精确匹配的数据来说是非常合适的，但是需要熟悉数据模型结构以构造相应的查询。

（7）分析方法。帮助更好地进行决策制定、数据挖掘和数据集成。分析组件与交互服务的组合应用程序的功能和特性紧密地结合在一起。并分析将构建增强的智能，以访问和关联来自异类信息源的信息，以使为更好地制定业务决策提供新的见解。

5．合作伙伴服务

合作伙伴服务用作为 SOA 的重用入口点，使其可以连接到 SOA 企业体系结构，并与 ESB 联系在一起，从而提高操作效率和 QoS；为每个后端系统或者业务应用程序都需要一个特定的适配器。而业务集成适配器由一组软件 API 组成，提供了与后端企业信息系统（EIS）的本地通信，以及配置业务对象和适配器的工具，如图 3-34 所示。

6．业务应用程序服务

业务应用程序服务构成了 SOA 的重用入口点。业务应用程序是松散耦合的，以使通过使用 Web 服务为企业带来业务价值。通过 Web 服务减少了构建昂贵的业务应用程序的成本，并且支持在企业结构中部署新的业务模型。对于大多数组织，在快速部署到主流的过程中，主要的问题是安全性。而业务应用程序服务合并了一些业务安全特性，以确保业务事务执行期间的安全。图 3-35 所示显示了使用业务流程和策略管理组件为企业的业务应用程序提供业

务安全服务的业务应用程序服务。

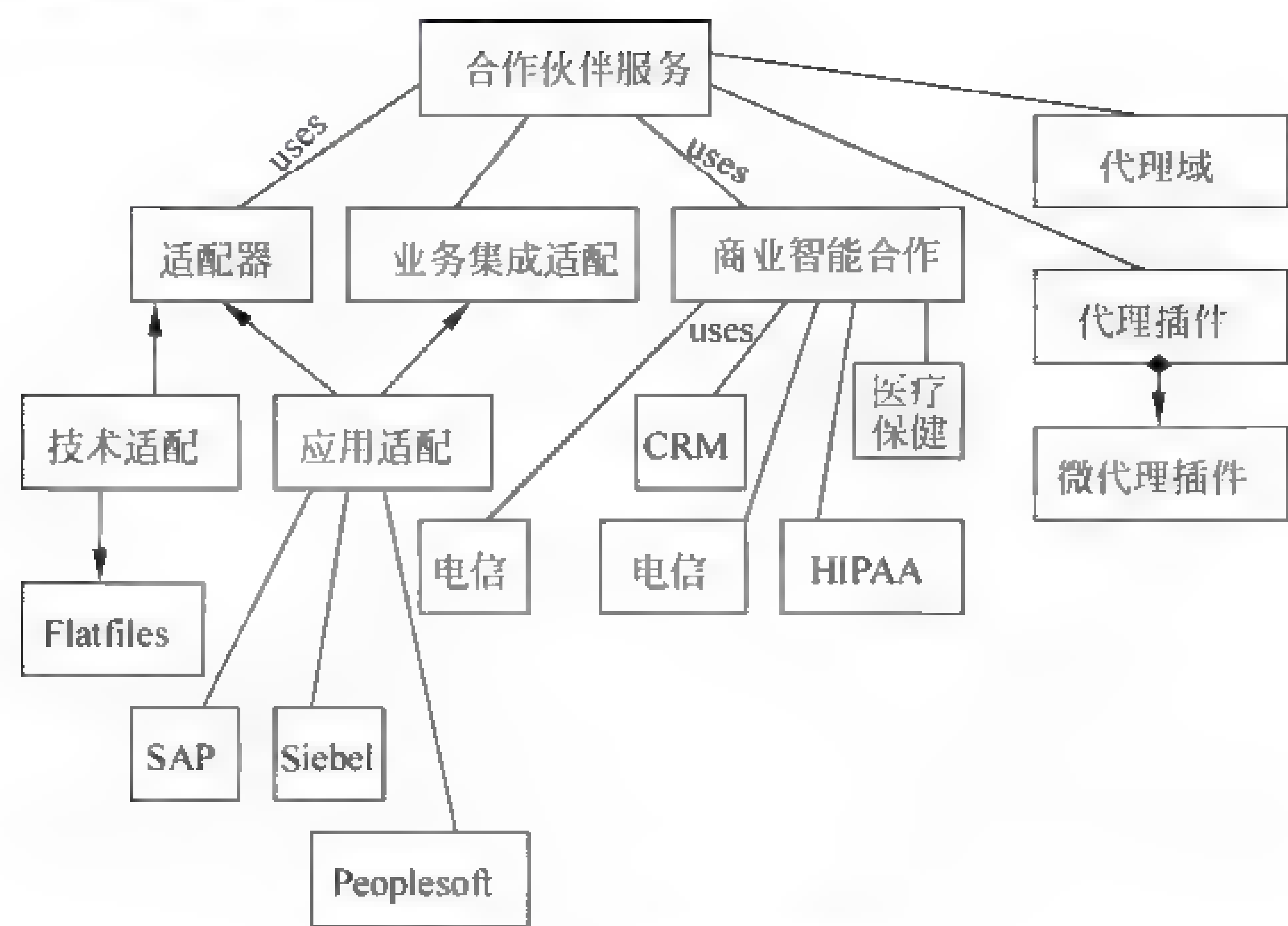

图 3-34　合作伙伴服务

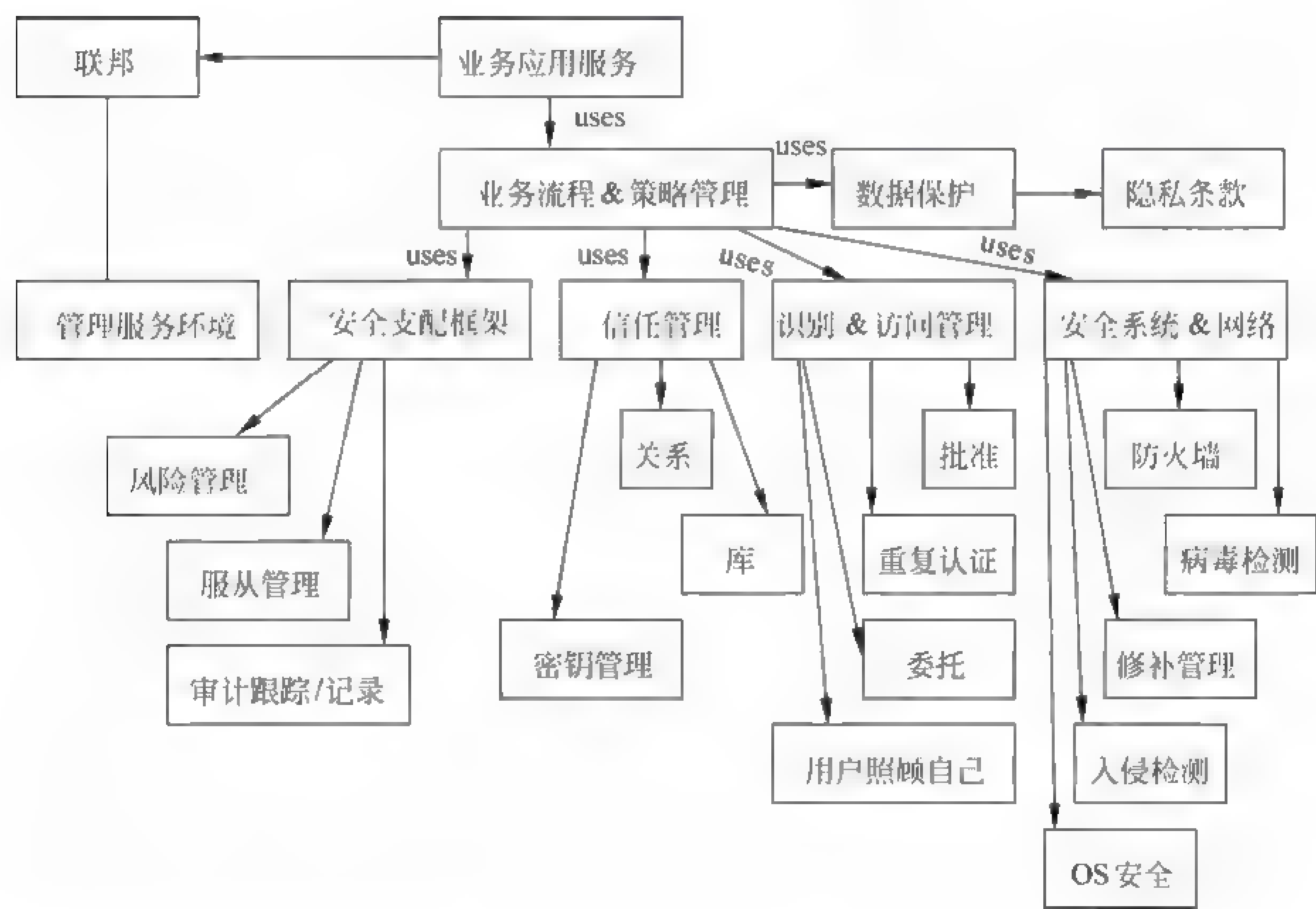

图 3-35　业务应用程序服务

7．应用程序和数据访问服务

应用程序和数据访问服务组件用作信息和 SOA 的重用入口点。图 3-36 所示显示了一个带有应用程序和数据访问服务组件的企业应用程序场景，它支持各种交互协议和 QoS。当大多数组织决定将应用程序公开为 SOA 环境中的服务时，它们的业务应用程序必须能够处理各种不同的数据表示形式。

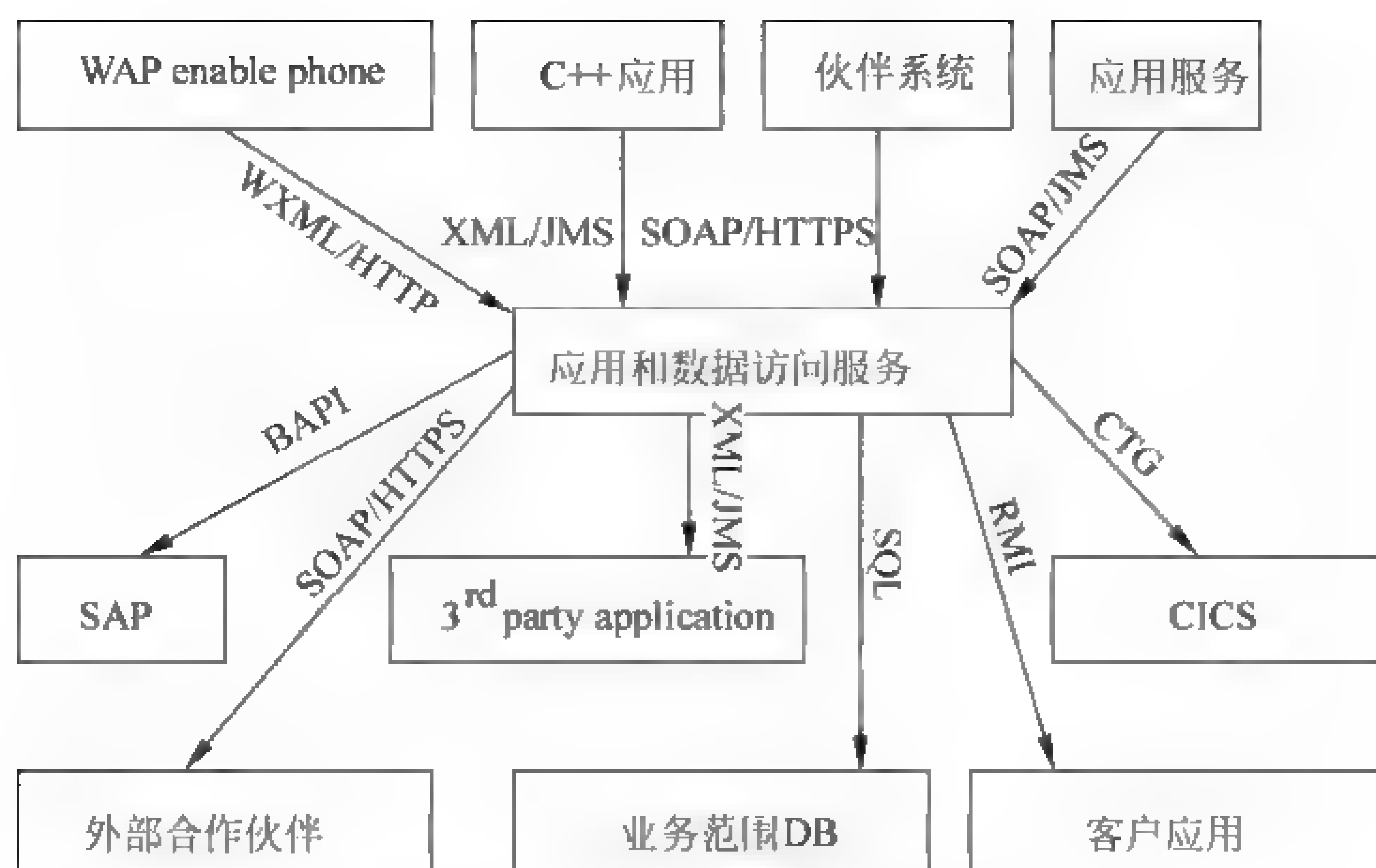

图 3-36　应用程序和数据访问服务

应用程序可以通过 SCA 编程模型公开与基础数据层进行交互的服务。而且还有一种称为关系数据库数据访问服务（RDB DAS）的可靠数据访问实用工具，它在基于 SCA 的应用程序中提供了与服务数据对象（SDO）的紧密集成，从而实现服务数据与应用程序的交互。

3.3.2.6　UML 与 SOA 转换

通常 UML 与 SOA 转换是以基于 SCA 为基础，并使用 Service Component Definition Language（SCDL）来表示元数据，实现 UML 与 SOA 映射及代码转换。典型工具是 IBM® Rational® Software Architect（IBMSA）中的 UML-to-SOA 的转换工具包，通过 UML-to-SOA 转换工具将生成准备导入到 IBM® WebSphere® Integration Developer（IBMID）6.0.2 或之后版本中，用于进一步开发、测试，和部署的输出，这样就可以实现 UML 与 SOA 的转换。

1. UML 与 SOA 转换原理

在 IBM 提供的工具中，将 IBMSA 中的结果导入到 IBMID 中实现基于 UML 的 SOA 软件研发。UML-to-SOA 转换的预期的源是一个完整的应用或没有应用 UML 2.0 Profile for Software Services 的 UML 模型。也就是说，转换可以将一个或多个表示服务提供者的 UML 包或个别的 UML 组件作为源。如果将 UML 模型或包选为源，那么 UML-to-SOA 转换就导航到所选的模型或包，并为每个服务提供者创建必要的 SCDL 工件。并且 UML-to-SOA 转换支持选择多个 UML 元素作为转换的源。同时，而对于 UML 与 SOA 转换输出则是导入到 IBM WebSphere Integration Developer 6.0.2 和之后版本中用于进一步开发、测试和部署，转换可以接受工作区中任意工程为目标。其在转换过程主要完成以下几部分的数据处理，源与目标输出的映射关系如表 3-7 所示。

（1）处理对现有 XSD 数据类型的引用。在对 UML 与转换过程中，软件服务的 UML 模型可能使用对现有 XSD 类型的引用。在这种情况下，UML-to-SCDL 转换创建一个库工程，并且将包含被引用的 XSD 数据类型的 XSD 资源从其父工程中拷贝到库工程中，包括文件夹结构和所有用 XSD include 或 XSD import 引用的 XSD 方案。创建的库工程的名字与包含最初被引用的 XSD 方案的工程的名字相同。如果被包含或被导入的 XSD 方案在不同的工程中，那么转换将创建相应的库工程，并且在需要处添加工程依赖。

（2）处理对 WSDL 端口类型的引用。在对 UML 与转换过程中，软件服务的 UML 模型用与现有的 XSD 数据类型相似的方式使用对现有 WSDL 端口类型的引用。转换对 WSDL 端口类型的引用的处理方式与对 XSD 数据类型的一样。利用通过 XSD include 或 XSD import 引用的 WSDL 导入和 XSD 方法，转换为在 UML 模型及所有被引用的 WSDL 文件中引用的 WSDL 端口类型创建所有必要的库工程。

（3）处理引用的 Java 接口和 Java 类。UML-to-SOA 转换通过引用现有 Java 接口和类作为源来支持软件服务的UML 模型。就能够将新的服务设计为已经拥有Java实现并且通过 Java 接口来显露现有服务的聚集或组合。当转换在解析源模块的过程中遇到 Java 接口或类时，它创建 Java 工程的副本，副本包含 Java 源代码，以及它所依赖的所有 Java 工程，这些被转移到被选为转换目标容器的工程中。

表 3-7 源和输出对象之间的映射

源	输出
拥有至少一个所提供的接口的 UML 组件	WID 模块工程 SCDL 模块
拥有至少一个所提供的接口的 UML 组件，其所拥有的行为是作为 UMLActivity	WID 模块工程 SCDL 模块工程 实现 BPEL 的 SCDL 组件
通过 UML 展示出的所提供的接口 表示服务提供者的 UML 组件的端口	SCDL 导出
通过表示软件服务的 UML 组件的 UML 端口展示出的所需的接口	SCDL 导入
表示软件服务的 UML 组件的 UML 部件	SCDL 组件
内部或外部端口所引用的所提供的接口	SCDL 接口 WSDL 接口
内部或外部端口所引用的所需的接口	SCDL 引用 WSDL 接口
所引用的接口的 UML 方法	SCDL 方法
UML 接口或 WSDL PortType 所引用的数据类型	XSD 数据类型
UML 连接器	SCDL 线

2. UML 与 SOA 转换实现

如图 3-31 是 UML-to-SOA 转换的结构图，在转换过程是由 SCDL 来实现的，目前 SCDL 只是一种 SCA 定义语言，以 IBM 应用最突出的企业之一，尚未形成一种 SCA 业界的标准的描述语言。其定义结构如表 3-8 所示。

表 3-8 SCDL 定义 SCA 模块

序号	SCA 项	SCDL 描述
1	模块定义	包含在 SCA 根据项目 JAR 的 sca.module 文件中
2	服务组件	（1）一个模块能包含 0...n 个服务定义 （2）每个组件被包含一个 <SERVICE_NAME>.component 文件
3	输入（Imports）	（1）一个模块能够包含 0...n 个服务定义 （2）每个输入定义包含一个 IMPORT_NAME>.import 文件
4	导出（Exports）	（1）一个模块能包含 0...n 个导出定义 （2）每个导出定义被包含一个<EXPORT_NAME>.export 文件

续表

序号	SCA 项	SCDL 描述
5	引用（References）	（1）包含内联（包含一个服务组件定义）和独立（Stand-alone）两个类型引用 （2）每个组件定义被包含一个 <SERVICE NAME>.reference 文件
6	其他 SCA 项	主要包含 Java™ Classes, WSDL files, Other Artifacts XSD files, BPEL

模块定义的描述语言定义：

```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:module xmlns:scdl=" SCDL 的 URI "
name=" 描述的名称"/>
```

服务组件描述语言定义：服务组件定义包括组件名称、实现、接口和引用，如图 3-37 所示。

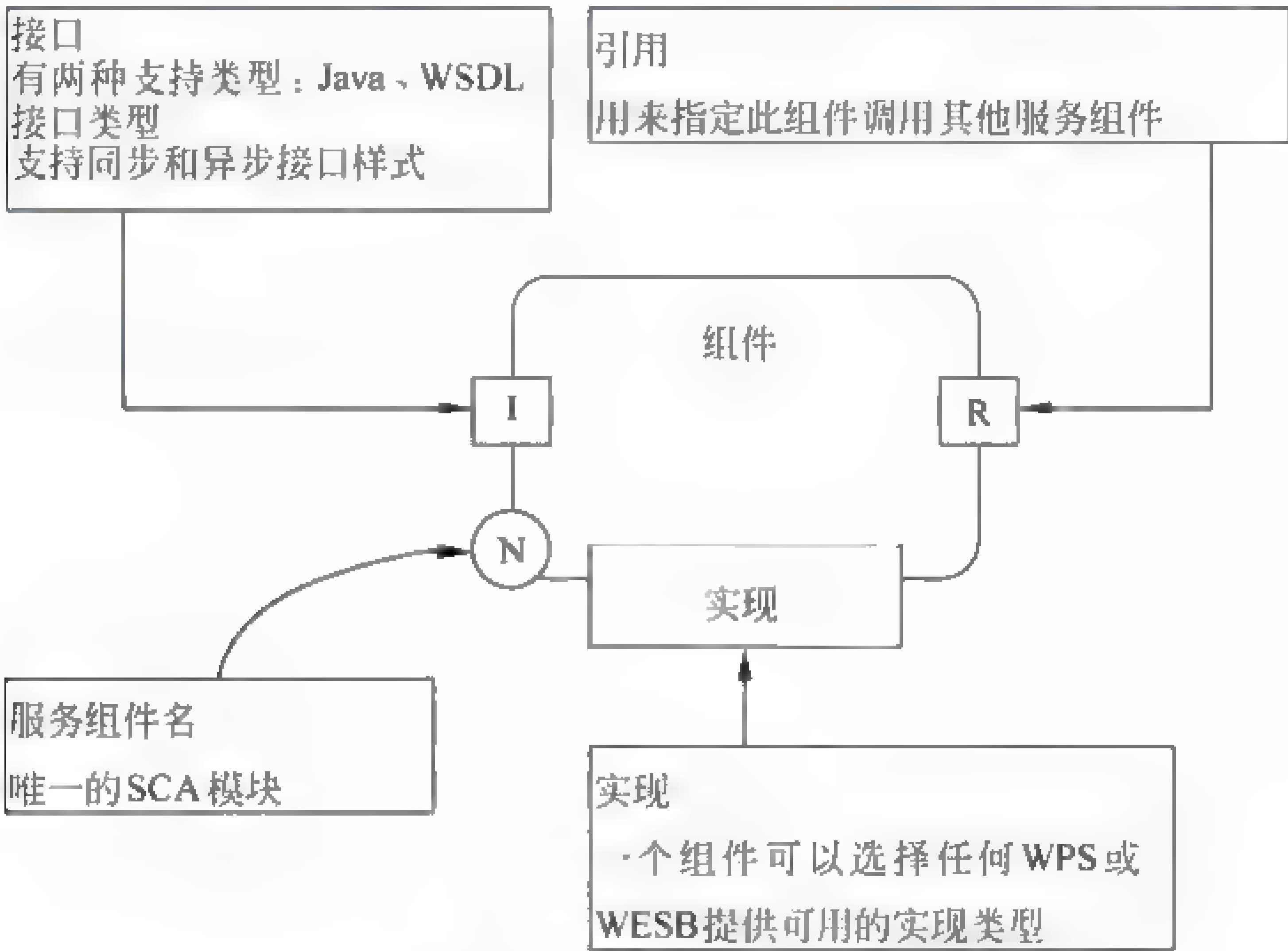

图 3-37　服务组件结构

下面例子是一个包含了一个 WSDL 接口定义、两个引用和一个实现的服务组件描述方法。

```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:component xmlns="http://www.w3.org/2001/XMLSchema-instance"
             xmlns:ns1="http://"
             xmlns:ns2="http://"
             xmlns:ns3="http://"
             xmlns:scdl="http://"
             xmlns:wsdl="http://"
             ......>
<interfaces>
<interface xsl:type="wsdl:WSDLPortType" portType="ns1:servicename1">
</interfaces>
<references>
```

```
<reference name " ">
  <interface xsl:type "WSDLPortType" portType "ns2:servicename2">
    <method name-""/>
  </interface>
  <wire target=""/>
</reference>
<reference name=" ">
   <interface xsl:type="WSDLPortType" portType="ns3:servicename3">
     <method name=""/>
   </interface>
     <wire target=""/>
   </reference>
</reference>
<implementation xsl:type="" mfcFile="">
</scdl:component>
```

下面例子是一个在服务组件中的输入接口定义方法：

```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:import xmlns="http://www.w3.org/2001/XMLSchema-instance"
             xmlns:ns1="http://"
             xmlns:scdl="http://"
             xmlns:webservice="htto://"
             xmlns:wsdl="http://"
             ......>
<interfaces>
<interface xsl:type="wsdl:WSDLPortType" portType="ns1:servicename1">
<method name=" "/>
</interface>
</interfaces>
<esbBinding xsl=type="webservice:..." endpoint="http://" port="ns1:..."
service="ns1:...">
</scdl:import>
```

下面例子是一个在服务组件中的导出接口定义：

```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:export xmlns="http://www.w3.org/2001/XMLSchema-instance"
             xmlns:ns1="http://"
             xmlns:scdl="http://"
             xmlns:webservice="htto://"
             xmlns:wsdl="http://"
             ......>
<interfaces>
<interface xsl:type-"wsdl:WSDLPortType" portType-"ns1:servicename1">
<method name-" "/>
```

```
    </interface>
    </interfaces>
    ......
    </scdl:export>
```

图 3-38 是包括内联和独立的 SCA 引用的结构图，语言描述结构同图 3-37 所示的定义方法。

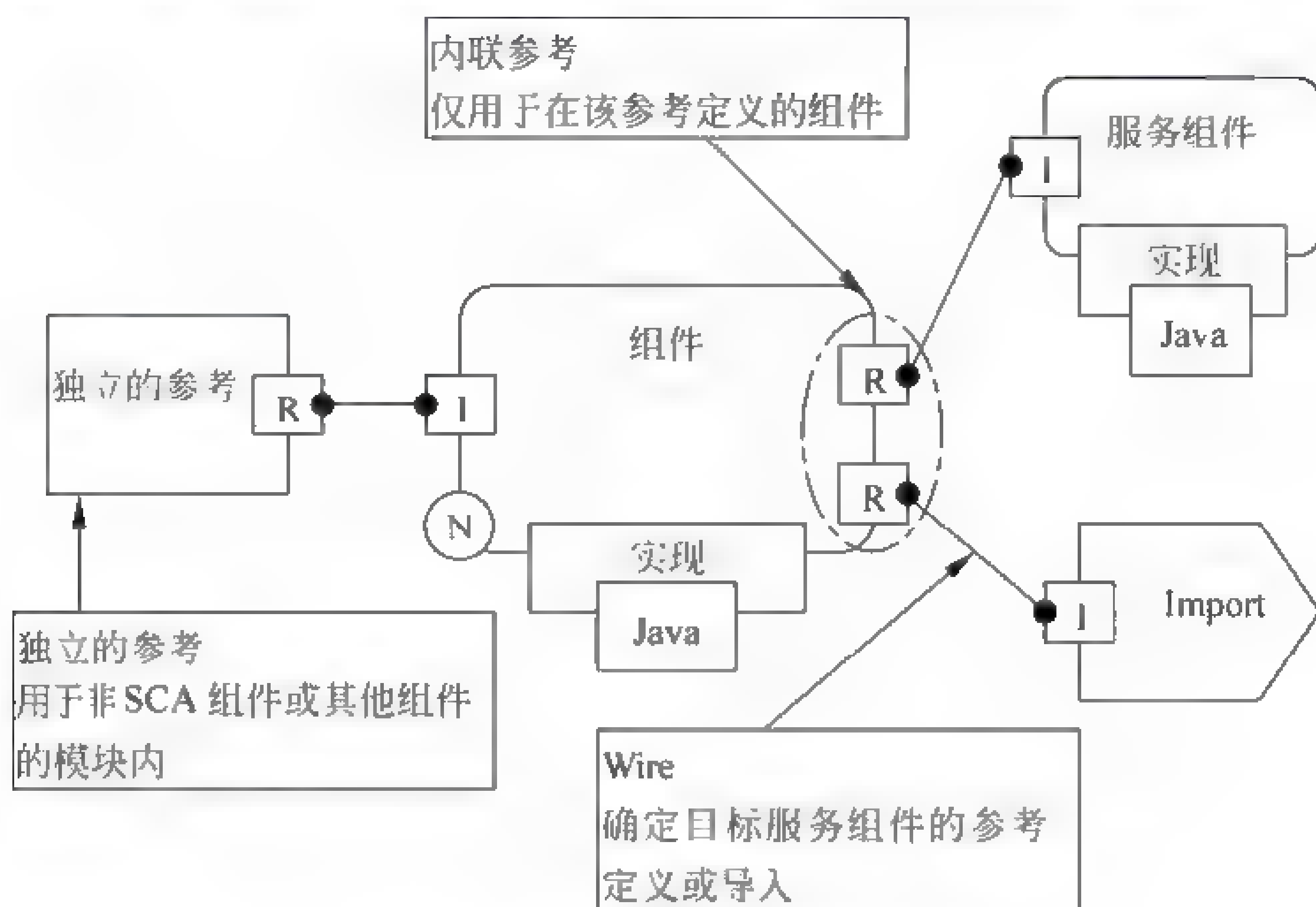

图 3-38 是 SCA 引用结构图

UML-to-SOA 转换工具是由 UML-to-SCDL 转换扩展而来，它负责创建 SCDL 模块和库工程。并且 UML-to-BPEL 转换工具使用 UML-to-WSDL 转换和 UML-to-XSD 转换来创建 WSDL 端口类型和 XSD Schema Datatypes；同样地，UML-to-WSDL 转换使用 UML-to-XSD 转换来创建 XSD Schema Datatypes，如图 3-39 所示。

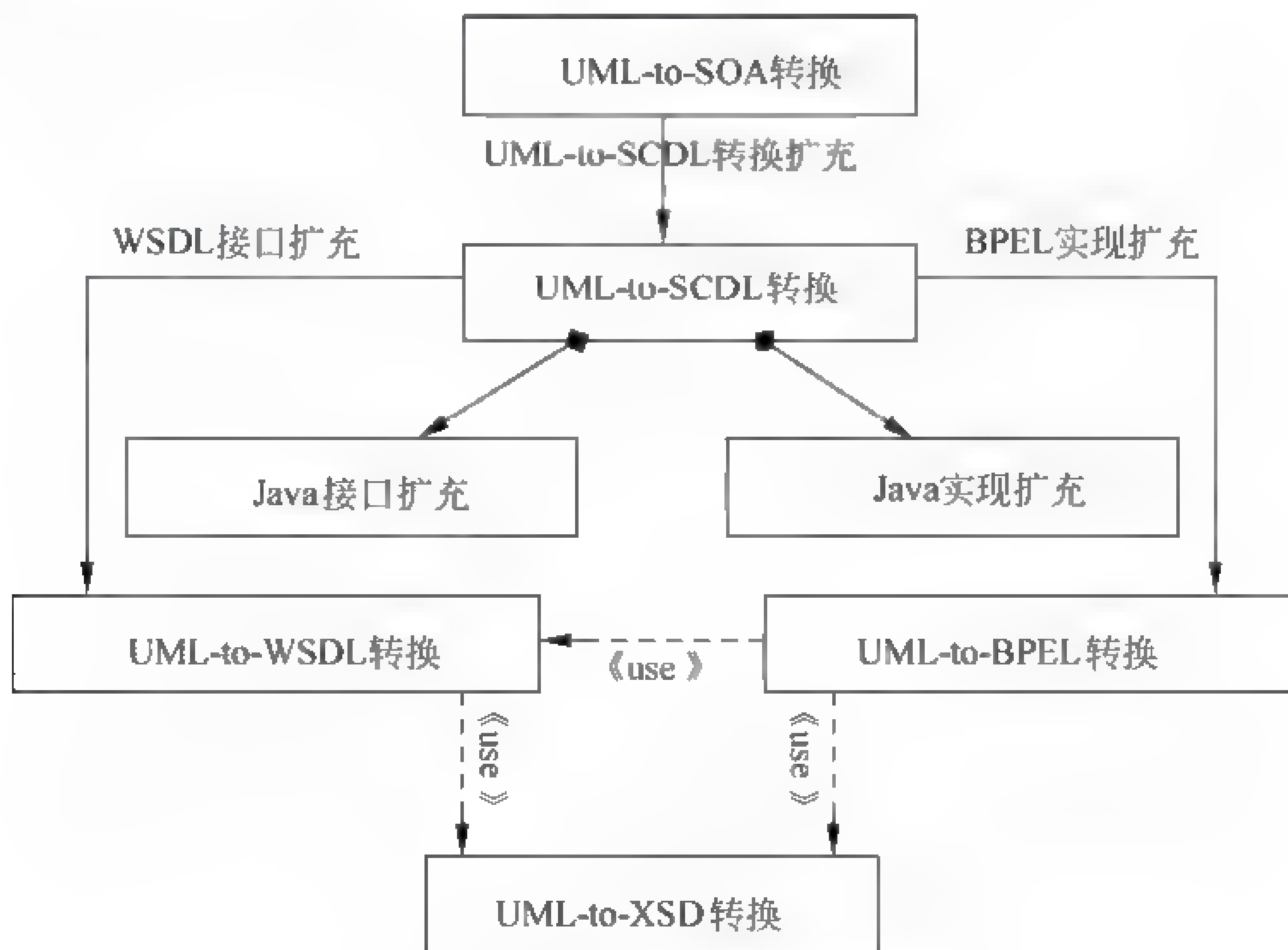

图 3-39 UML 与 SOA 转换结构

其主要内容如下：

（1）模块工程包含 SCDL 工件，例如模块、组件、导出和导入。

（2）库工程包含由各种 SCDL 元素所引用的其他具体领域的工件。

这时在 IBMSA 中包含如下扩展接口用于 UML-To-SCDL 转换：

① 接口扩展。用于处理源模型中的所有接口，包括 UML 接口、所引用的 WSDL 端口类型和 Java 接口。WSDL 接口扩展使用 UML-to-WSDL 转换来创建 SCDL 组件、导出和导入所引用的 WSDL 端口类型。Java 接口扩展用于将包含所引用的 Java 接口的 Java 工程复制到目标工程中。

② 组件实现扩展。用于处理 UML 活动和对 Java 类的引用。Business Process Execution Language（BPEL）实现扩展使用 UML-to-BPEL 转换来创建 SCDL 所引用的 BPEL 过程。Java 实现扩展用于将包含所引用的 Java 类的 Java 工程复制到目标工程中。

3.3.3 面向服务流程的建模方法

一般认为业务流程可以视为业务实体为了响应事件而执行的一组活动。这一组活动在业务流程中彼此协调，一起描述并集成到一起。而在 SOA 范式中，业务流程对服务流进行控制。业务流程驱动事件流、调用和协调服务，并为其创建上下文来进行相互通信。业务流程代表业务抽象，同服务实现进行了分离，是关于业务流的流程。通过这个关注分离，不仅可以将更多精力放在流程创建上，而且还可以更方便地根据需要编辑流程，但不用对基础服务实现进行编辑。通常业务流程是以特定顺序调用以实现业务目标的一组业务相关的活动。业务流程由多项任务组成，这些任务包括：人工交互、自动化、工作流、信息服务、业务规则交互、子流程、调用功能和服务。而以下主要从服务流程的基础和 UML 与 BPEL 映射方法展开论述。图 3-40 是流程与服务的结构图。

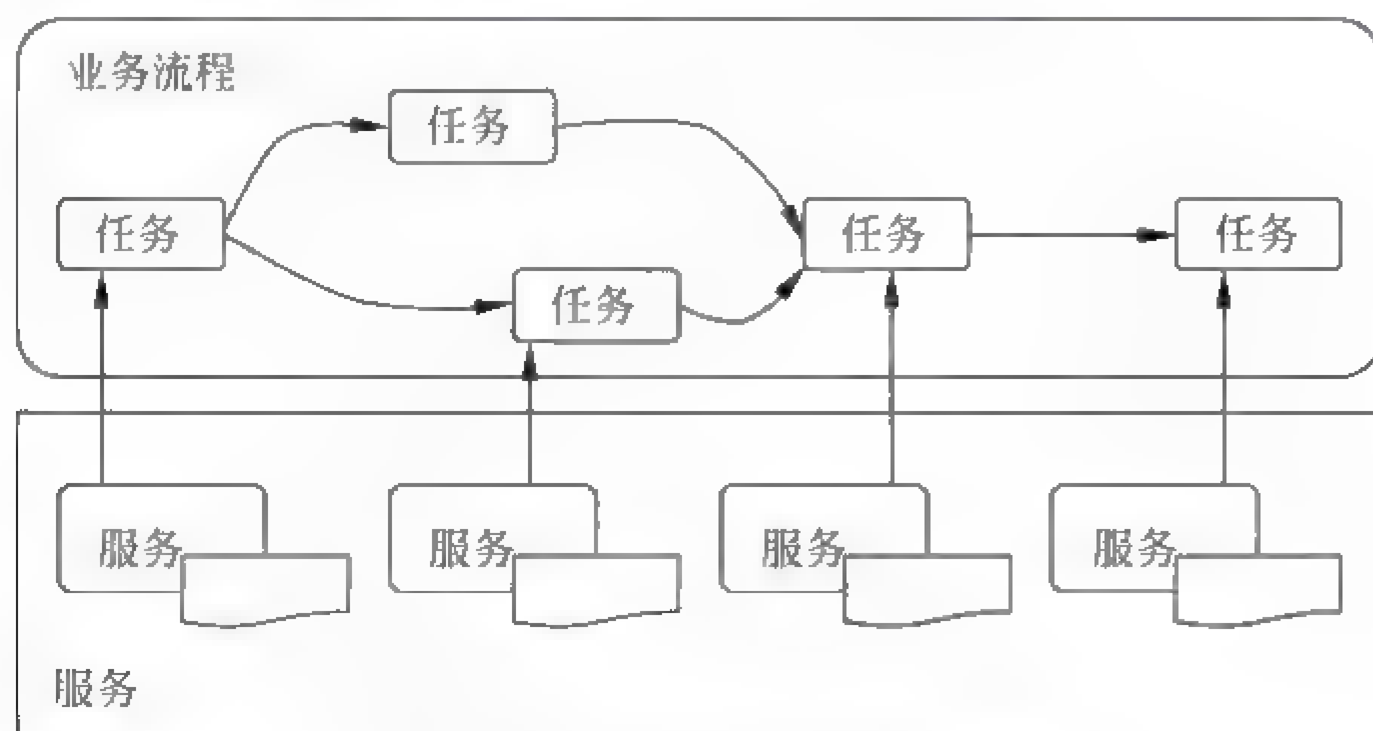

图 3-40 流程与服务的结构图

3.3.3.1 BPEL 发展简介

2002 年 7 月，IBM、微软、BEA 提交了 Business Process Execution Language for Web Services（BPEL4WS）1.0 的规范。业务流程执行语言是基于 XML 和 Web 服务的技术，它融合了早期的 IBM 的 Web Services Flow Language（WSFL）及微软的 XLANG 规范的很多特点。随后许

多主要供货商如 SAP 和 Siebel（已被 Oracle 并购）等公司陆续加入规范的制定，并催生了多项修改和改进，并于 2003 年 3 月发布了 1.1 版。2003 年 4 月，BPEL 被提交结构化信息标准促进组织（OASIS）以实现标准化，并组建了 Web 服务业务流程执行语言技术委员会（WSBPEL TC），努力使 BPEL 在业界获得更为广泛的认可。目前该技术委员会正在致力于下一代规范的制定工作，并将该规范重命名为 WS-BPEL 2.0。虽然除 BPEL 之外还有一些业务流程规范，但是到目前为止，BPEL 是最为成熟和被广泛支持的技术。

3.3.3.2 服务业务流程的基础

OASIS 标准组织已将 Business Process Execution Language（BPEL）定义为基于标准的方法，使用该方法可以编排由服务构成的业务流程。2007 年，WS-BPEL 2.0 被批准为标准语言。作为一种执行语言，WS-BPEL 定义了如何表示业务流程中的活动，以及流控制逻辑、数据、消息相关性和异常处理等。

1. 业务流程的组成元素

从组成元素的角度定义业务流程的做法可能相对更好一些，这样能从技术角度对业务流程有一些了解。

（1）输入。流程的活动为了产生结果所需的信息。如在身份证示例中，输入应该为凭据、出生证明和照片。

（2）输出。流程生成的所有数据和信息。输出代表业务目标和业务所需的度量数据。如在身份证示例中，输出的是内部文件和实际身份证以及关于流程如何进行的度量数据。

（3）事件。一些重要情况的通知。例如，某个指示信息，事件可能在流程的执行前、执行中和执行后发生。如在身份证示例中，可能会出现关于最开始没有提供需要包括新文档的事件。

（4）子流程。流程中较小的流程或流程步骤。使用一组活动不足以表示工作范围时，将使用子流程。子流程具有与流程相同的组成元素。如在身份证示例中，这可以为调查犯罪记录并获得结果的子流程。

（5）活动。流程中级别最低的工作单位。如在身份证示例中，为申办身份证的人创建新内部文件的工作就是一个活动。

（6）性能度量。表示流程有效性的属性，用于确定是否满足所需的性能。这些度量可以帮助确定性能并将其与所需的指标进行比较。通过这些性能度量还能确定流程改进的潜在区域，最终实现 SOA 承诺的改进周期。如在身份证示例中，度量数据中将计算流程的哪个部分使用的时间最多或处理命中率最高。

2. BPEL 的基本特性

BPEL 相对于对象组装技术，服务组装更为复杂。人们必须面对 SOA 环境中异构的、松耦合的和自主的服务。它们间的交互关系是动态的、按需发生的，而且缺少中央控制。因此，BPEL 提供的服务组装模型提供了下列特性：

（1）灵活性。服务组装模型应该具有丰富的表现能力，能够描述复杂的交互场景，而且能够快速地适应变化。

（2）嵌套组装。一个业务流程可以表现为一个标准的 Web 服务，并被组装到其他流程或

服务中，构成更粗粒度的服务，提高了服务的可伸缩性和重用性。

（3）关注点分离。BPEL 只关注与服务组装的业务逻辑和其他关注点，比如服务质量（QoS，Quality of Service），事务处理等，可被作为附加扩展，由具体实现平台进行处理。

（4）会话状态和生命周期管理。与无状态的Web服务不同，一个业务流程通常具有明确的生命周期模型。BPEL 提供了对长时间运行的、有状态交互的支持。

（5）可恢复性。这对于业务流程（尤其对长时间运行的流程）是非常重要的。BPEL 提供了内置的失败处理和补偿机制，对于可预测的错误进行必要的处理。

3. WS-BPEL 介绍

使用正式和标准的语言来描述业务流程，这个观念由来已久。到目前为止，有许多可用于完成此项任务的竞争性标准，包括IBM的Web服务流语言（WSFL）、Microsoft的XLANG，以及许多其他标准。最后，结合WSFL和XLANG的优点而创建了WS-BPEL。在这类标准中，它已成为最流行的标准。

WS-BPEL 依赖于各种各样的标准技术，具体包括：WSDL、XML 模式、Xpath 和 Web服务寻址。其中 WSDL 是这些标准中最为重要的，因为 WS-BPEL 将业务流程描述为 Web 服务间的会话，而这些Web服务使用WSDL进行描述。此外，WS-BPEL 流程是 WSDL 描述的Web服务。

同样，WS-BPEL 是一种XML编程语言。在 WS-BPEL 中，可以在业务流程执行环境所执行的XML文档中对业务流程进行描述。WS-BPEL中的一个流程可以分解为一系列称为活动的操为步骤。所支持的活动的列表如下所示：

（1）invoke——在现有的 Web 服务上调用一项操作，Invoke 活动使用"partnerLink"来引用伙伴服务，通过"portType"和"operation"指定相应的WSDL接口和操作：

```
<bpws:invoke name=" " operation=" " partnerLink=" " portType="URI" >
</bpws:invoke>
```

（2）receive——等待来自外部实体的消息。

（3）reply——产生一条应答外部实体的消息，可以描述如下：

```
<reply name="reply"
      partnerLink=" "
      portType=" "
      operation=" "
      variable=" ">
</reply>
```

（4）wait——等待一段指定的时间。

（5）assign——将一个值从源复制到目标，<assign>活动的作用是用新的数据来更新变量的值，Assign活动可以包括任意数量的基本复制操作，并且assign活动还可把端点引用复制到合作伙伴链接，或把合作伙伴链接复制到端点引用，以实现服务的动态绑定。其表示如下：

```
<bpws:assign name="Assign">
  <bpws:copy>
    <bpws:from> China</bpws:from>
```

```
    <bpws:to variable "country"/>
  </bpws:copy>
</bpws:assign>
```

（6）throw——引发一个错误。

（7）terminate——无条件地中止当前流程。

（8）empty——提供一个 no-op 占位符。

可以使用一组附加的语言关键字，对这些基本活动进行组合，以描述业务流程算法，这些关键字提供了 WS-BPEL 语言的结构特性，下面分别列出了这些关键字。

（1）sequence——定义一组操作的有序序列，可以描述如下：

```
<sequence>
  <receive name=" " … >
  </receive>
<!-- Do something interesting -->
 </sequence>
```

（2）switch——提供选择语句，其工作方式类似于 Java 和 C++ 中的 case。switch 在 BPEL 中的描述方法如下：

```
<switch>
  <case condition=" 判断条件 ">
    <invoke name="invokeClassified"
            partnerLink=" "
            portType="  "
            operation="  "
            inputVariable=" "
            outputVariable=" ">
    </invoke>
  </case>
  <otherwise>
    <assign name="assignMessage">
      <copy>
        <from expression="' '"/>
        <to variable=" " part="accept"/>
      </copy>
    </assign>
  </otherwise>
</switch>
```

（3）while——定义一个 while 循环。

（4）pick——提供选择语句，其工作方式类似于 if。

（5）flow——封装一组应该并行执行的操作步骤。

使用这组关键字，可以在 WS-BPEL 中表示业务流程。

3.3.3.3　UML 与 BPEL 映射方法

随着 SOA 的出现和应用，应用程序开发经历了一次彻底改变。这种架构合并了基于 XML

的标准的 WSDL、简单对象访问协议（SOAP）、统一描述、发现和集成协议（UDDI）和现在的 BPEL 等。然而，当在分布式环境中使用软件系统做更多的事情时，并随着信息量日益增加和用户群体增长时，开发任务的大小和复杂性已经提高，并且开发者会发现他们沉迷于文档和语法中，而迷失了他们工作的主要目标。另外，开发人员的目标也正在发生变化，相应的软件技术标准本身也处在发展之中。因而，为了能快速的采用 Web 服务，开发者正在寻找解决复杂、高效和技术改变问题的答案。因此，业务流程执行语言（BPEL）是工业界在 Web Service 诸多规范的和目前所遇到问题的基础上提出的一种新型的流程描述语言，其初衷是为了更好地解决企业 IT 系统整合所面临的诸多问题。

BPEL 提供了一种 XML 注释和语义，用于指定基于 Web 服务的业务流程行为。使用合作伙伴的交互方式，定义了该 BPEL4WS（Business Process Executable Language for Web Service）流程。合作伙伴可以将服务提供给流程，也可以向流程请求服务，或者参与到流程的双向交互中。BPEL 通过指定顺序来编排 Web 服务，这对服务集合的调用来说意义深远。BPEL 还针对每个服务分配了合作伙伴的责任。从而可以使用它来指定合作伙伴的公共接口和可执行流程的描述。

1．BPEL 基本原理

BPEL 是一种基于 XML 的工作流定义语言，它使企业能够描述既能使用又能提供 Web 服务的复杂的业务流程。它最初是由 Microsoft、IBM 和 BEA 共同开发，它融合了 Microsoft 和 IBM 各自开发的上一代流程语言：XLANG 和 WSFL。BPEL 的基本功能在于能够对 Web 服务加以编排和协调，以便它们开展协作和事务性行为。BPEL 规范已作为协议标准提交至 OASIS 标准机构进行审核和最终命名，以供公众使用。并且 BPEL 通过采用 XML Schema 编写，从形式上它定义了用于组成复杂的业务流程交互的基础和结构化活动，指定了业务流程是怎样使用 Web 服务来达到它的目的以及由业务流程提供的 Web 服务。BPEL 是一种以 XML 来描述企业内部流程的语言，使原本建立在不同产品上的商业流程也能像 Web 服务一样可以跨平台互通。

业务流程执行语言（BPEL）是工业界在 Web Service 诸多规范的基础上提出的一种新型的流程描述语言，其初衷是为了更好地解决企业 IT 系统整合所面临的诸多问题。在当前激烈竞争的企业环境下，企业必须能够对于外界环境变化做出更快的反应，其中一种重要的适应策略就是改变和优化企业运行的业务流程。从技术支撑的层面来看，需要从流程建模、流程引擎和流程监测三个方面提供必要的支持。其中，流程建模帮助业务人员规约流程在功能层面和质量属性层面的基本特征，流程引擎负责管理和维护业务流程实例的生命周期，流程监测则为分析和评估业务流程的运行成本提供了可能，以上三个方面构成了一个闭环的回馈系统。当然 BPEL 作为一种新型的流程描述语言，是 IT 技术在流程建模层面的具体体现，它为基于流程的业务整合奠定了基础。

从编程语言的角度来看，BPEL 定义了一套完整的程序执行原语，如顺序、并发、分支和循环等，提供了一套类似于传统编程语言的概念集合，所以使用 BPEL 描述的流程可以作为运行实体直接部署于流程引擎，如图 3-41 所示。从业务流程抽象描述的角度来看，BPEL 支持在流程活动与用户角色之间建立映射，这一机制使得运行引擎在自动记录业务流程运行状态的同时，间接记录了用户在业务层面的操作行为，所以 BPEL 为流程监测奠定了基础。在 BPEL 提出之前，工业界也提出过若干种流程描述语言和开发环境，如 FDML、SAP Flow

和 FileNet 等，但是以上流程语言很难解决通信协议及访问接口之间的异构问题。BPEL 与以前诸多流程描述语言的最大不同之处在于 BPEL 对参与流程执行的协作方所采用的交互协议做出了明确的规定，即基于 XML 的 SOAP 编码。所以，BPEL 在信息集成领域得到了广泛的应用。

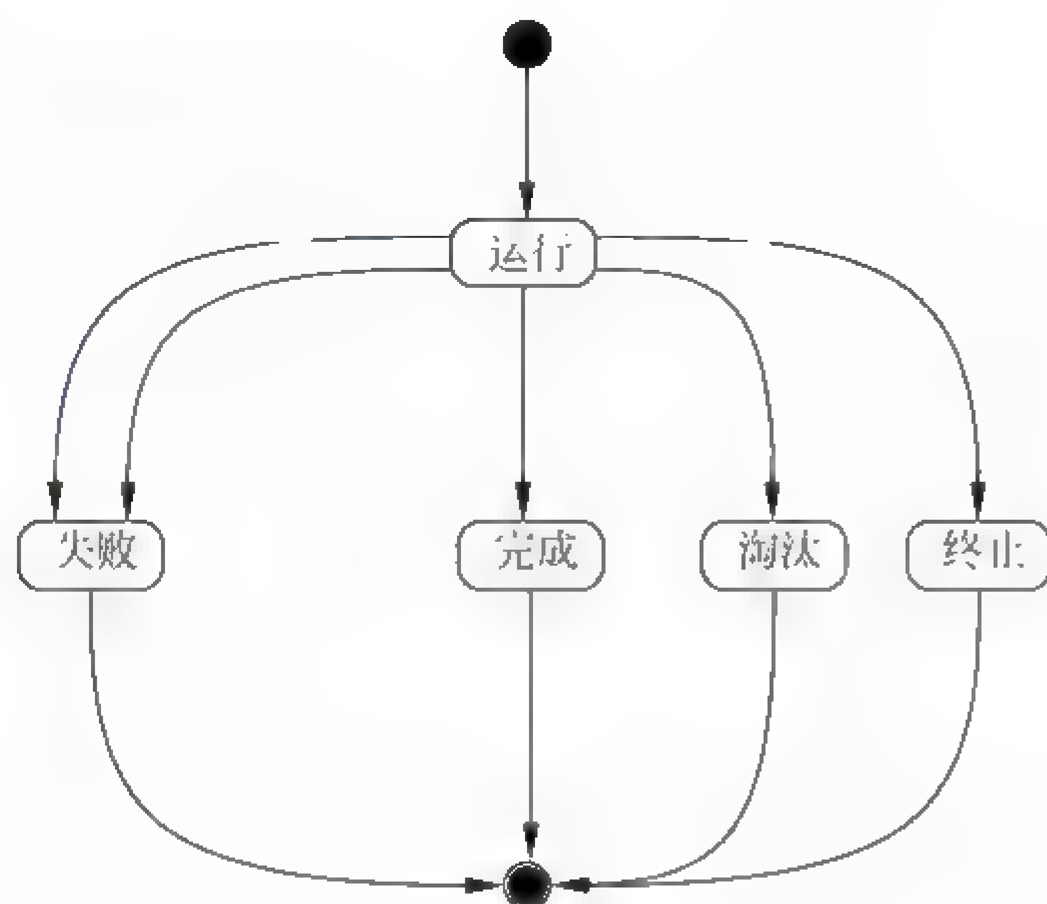

图 3-41　BPEL 流程活动状态转换示意图

然而，现实世界中的业务流程不仅复杂，而且结合了数不清的完整性控制：ACID（原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation，又称独立性）、持久性（Durability）事务支持、长期运行的交互过程的状态持久性、嵌套和并行操作、补偿和例外机制、确认和关联能力。随着流程的复杂性与日俱增和用户要求越来越高，能够使用可视化的组装方法设计、实现和归档高度独立和复杂行为的能力所具有的价值也日益被人们所认识。尽管 Web 服务为应用程序通过无边界网络相互传递消息和调用方法提供了一种方法，但是它们仍然不能利用自身的力量满足业务流程的操作需求。业务流程由一组独立和有序的操作组成，每个操作的执行结果都是适时、可预期和可重复的。通过编排和协调，使得 Web 服务能够满足这些需求。当前编排和协调 Web 服务有多种方式，如面向 Web 服务的业务流程执行语言（Business Process Execution Language for Web Services，BPEL4WS 或 BPEL）、Web 服务编排接口（Web Services Choreography Interface，WSCI）、业务流程建模语言（Business Process Modeling Language，BPML）、Web 服务对话语言（Web Services Conversation Language，WSCL）、Web 服务流语言（Web Services Flow Language，WSFL）等。

业务流程 process 是元模型中最重要的元素，描述了业务流程模型的结构和控制信息。通常情况下，BPEL 业务流程接收请求。为了满足请求，该流程调用相关的 Web 服务，然后响应原始调用方。由于 BPEL 流程与其他 Web 服务通信，因此它在很大程度上依赖于复合型 Web 服务调用的 Web 服务的 WSDL 描述。BPEL 元模型元素依赖于 WSDL 元素如类型端口（Port types）、消息（Messages）、操作（Operations）。

下面以一个商品服务为例来说明 EBPL 的使用方法。商品服务由商品发货和商品订购服务组成，并且商品服务提供两种选择，一种是让所发货的商品装载在一起，另一种是实现商

品分批订购和装载[1]。

（1）服务描述。下面程序是商品服务的描述，即它就是一个客户和服务相互作用的代码，用一个 partnerLinkType 来定义说明商品服务模型，命名为 shippingLT.wsdl，其程序清单如下：

```
<wsdl:definitions
targetNamespace="http://example.com/shipping/partnerLinkTypes/"
xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
xmlns:sif="http://example.com/shipping/interfaces/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
<wsdl:import location="shippingPT.wsdl"
namespace="http://example.com/shipping/interfaces/" />
<plnk:partnerLinkType name="shippingLT">
<plnk:role name="shippingService"
portType="sif:shippingServicePT" />
<plnk:role name="shippingServiceCustomer"
portType="sif:shippingServiceCustomerPT" />
</plnk:partnerLinkType>
</wsdl:definitions>
```

相应的 message 和 portType 定义描述如下，命名为 shippingPT.wsdl。

```
<wsdl:definitions
targetNamespace="http://example.com/shipping/interfaces/"
xmlns:ship="http://example.com/shipping/ship.xsd"
xmlns:tns="http://example.com/shipping/interfaces/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<wsdl:types>
<xsd:schema>
<!-- import ship schema -->
</xsd:schema>
</wsdl:types>
<wsdl:message name="shippingRequestMsg">
<wsdl:part name="shipOrder" type="ship:shipOrder" />
</wsdl:message>
<wsdl:message name="shippingNoticeMsg">
<wsdl:part name="shipNotice" type="ship:shipNotice" />
</wsdl:message>
<wsdl:portType name="shippingServicePT">
<wsdl:operation name="shippingRequest">
<wsdl:input message="tns:shippingRequestMsg" />
</wsdl:operation>
</wsdl:portType>
<wsdl:portType name="shippingServiceCustomerPT">
```

① http://xml.coverpages.org/BPELv11-May052003Final.pdf

```
<wsdl:operation name="shippingNotice">
<wsdl:input message="tns:shippingNoticeMsg" />
</wsdl:operation>
</wsdl:portType>
</wsdl:definitions>
```

（2）与服务相关的属性：

商品订购 ID（shipOrderID）被用来关联订购商品通知；用 shipComplete 描述订购是否完成；用 itemsTotal 描述订购的商品的总数；当分批订购被接受时，itemsCount 表示订购通知的数目，itemsTotal 用于追踪订购实现或完成。将其命名为 shippingProperties.wsdl。其程序清单如下：

```
<wsdl:definitions
targetNamespace="http://example.com/shipping/properties/"
xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
xmlns:vprop="http://docs.oasis-open.org/wsbpel/2.0/varprop"
xmlns:ship="http://example.com/shipping/ship.xsd"
xmlns:sif="http://example.com/shipping/interfaces/"
xmlns:tns="http://example.com/shipping/properties/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<wsdl:import location="shippingPT.wsdl"
namespace="http://example.com/shipping/interfaces/" />
<!-- types used in Abstract Processes are required to be finite
domains. The itemCountType is restricted by range -->
<wsdl:types>
<xsd:schema
targetNamespace="http://example.com/shipping/ship.xsd">
<xsd:simpleType name="itemCountType"> <xsd:restriction base="xsd:int">
<xsd:minInclusive value="1" />
<xsd:maxInclusive value="50" />
</xsd:restriction>
</xsd:simpleType>
</xsd:schema>
</wsdl:types>
<vprop:property name="shipOrderID" type="xsd:int" />
<vprop:property name="shipComplete" type="xsd:boolean" />
<vprop:property name="itemsTotal" type="ship:itemCountType" />
<vprop:property name="itemsCount" type="ship:itemCountType" />
<vprop:propertyAlias propertyName="tns:shipOrderID"
messageType="sif:shippingRequestMsg" part="shipOrder">
<vprop:query>
ship:ShipOrderRequestHeader/ship:shipOrderID
</vprop:query>
</vprop:propertyAlias>
```

```
<vprop:propertyAlias propertyName "tns:shipOrderID"
messageType "sif:shippingNoticeMsg" part "shipNotice">
<vprop:query>ship:ShipNoticeHeader/ship:shipOrderID</vprop:query>
</vprop:propertyAlias>
<vprop:propertyAlias propertyName="tns:shipComplete"
messageType="sif:shippingRequestMsg" part="shipOrder">
<vprop:query>
ship:ShipOrderRequestHeader/ship:shipComplete
</vprop:query>
</vprop:propertyAlias>
<vprop:propertyAlias propertyName="tns:itemsTotal"
messageType="sif:shippingRequestMsg" part="shipOrder">
<vprop:query>
ship:ShipOrderRequestHeader/ship:itemsTotal
</vprop:query>
</vprop:propertyAlias>
<vprop:propertyAlias propertyName="tns:itemsCount"
messageType="sif:shippingRequestMsg" part="shipOrder">
<vprop:query>
ship:ShipOrderRequestHeader/ship:itemsCount
</vprop:query>
</vprop:propertyAlias>
<vprop:propertyAlias propertyName="tns:itemsCount"
messageType="sif:shippingNoticeMsg" part="shipNotice">
<vprop:query>ship:ShipNoticeHeader/ship:itemsCount</vprop:query>
</vprop:propertyAlias>
</wsdl:definitions>
```

（3）为了方便起见，下面的 BEPL 的定义不包含诸如完成流程描述的所提供错误条件的详细内容。下面是流程描述：

```
receive shipOrder
if
condition shipComplete
send shipNotice
else
itemsShipped := 0
while itemsShipped < itemsTotal
itemsCount := opaque // non-deterministic assignment
// corresponding e.g. to
// internal interaction with
// back-end system
send shipNotice
itemsShipped = itemsShipped + itemsCount
```

而 WS-BPEL process 描述如下：

```
<process name "shippingService"
```

```
targetNamespace "http://example.com/shipping/"
xmlns "http://docs.oasis-open.org/wsbpel/2.0/process/abstract"
xmlns:plt="http://example.com/shipping/partnerLinkTypes/"
xmlns:props="http://example.com/shipping/properties/"
xmlns:ship="http://example.com/shipping/ship.xsd"
xmlns:sif="http://example.com/shipping/interfaces/"
abstractProcessProfile="http://docs.oasis-open.org/wsbpel/2.0/process/abs
tract/ap11/2006/08">
<import importType="http://schemas.xmlsoap.org/wsdl/"
location="shippingLT.wsdl"
namespace="http://example.com/shipping/partnerLinkTypes/" />
<import importType="http://schemas.xmlsoap.org/wsdl/"
location="shippingPT.wsdl"
namespace="http://example.com/shipping/interfaces/" />
<import importType="http://schemas.xmlsoap.org/wsdl/"
location="shippingProperties.wsdl"
namespace="http://example.com/shipping/properties/" />
<partnerLinks>
<partnerLink name="customer" partnerLinkType="plt:shippingLT"
partnerRole="shippingServiceCustomer"
myRole="shippingService" />
</partnerLinks>
<variables>
<variable name="shipRequest" messageType="sif:shippingRequestMsg" />
<variable name="shipNotice" messageType="sif:shippingNoticeMsg" />
<variable name="itemsShipped" type="ship:itemCountType" />
</variables>
<correlationSets>
<correlationSet name="shipOrder" properties="props:shipOrderID" />
</correlationSets>
<sequence>
<receive partnerLink="customer" operation="shippingRequest"
variable="shipRequest">
<correlations>
<correlation set="shipOrder" initiate="yes" />
</correlations>
</receive>
<if>
<condition>
bpel:getVariableProperty('shipRequest','props:shipComplete')
</condition>
<sequence>
<assign>
<copy>
<from variable "shipRequest" property "props:shipOrderID" />
```

```
<to variable "shipNotice" property "props:shipOrderID" />
</copy>
<copy>
<from variable="shipRequest" property="props:itemsCount" />
<to variable="shipNotice" property="props:itemsCount" />
</copy>
</assign>
<invoke partnerLink="customer" operation="shippingNotice"
inputVariable="shipNotice">
<correlations>
<correlation set="shipOrder" pattern="request" />
</correlations>
</invoke>
</sequence>
<else>
<sequence>
<assign>
<copy>
<from>0</from>
<to>$itemsShipped</to>
</copy>
</assign>
<while>
<condition>
$itemsShipped
&lt;
bpel:getVariableProperty('shipRequest','props:itemsTotal')
</condition>
<sequence>
<assign>
<copy>
<opaqueFrom/>
<to variable="shipNotice" property="props:shipOrderID" />
</copy>
<copy>
<opaqueFrom/>
<to variable="shipNotice" property="props:itemsCount" />
</copy>
</assign>
<invoke partnerLink="customer" operation="shippingNotice"
inputVariable="shipNotice">
<correlations>
<correlation set="shipOrder" pattern="request" />
</correlations>
</invoke>
```

```
<assign>
<copy>
<from>
$itemsShipped+bpel:getVariableProperty('shipNotice','props:itemsCount')
</from>
<to>$itemsShipped</to>
</copy>
</assign>
</sequence>
</while>
</sequence>
</else>
</if>
</sequence>
</process>
```

从以上 BPEL 代码可以得到以下信息：BPEL 使用<process>来定义流程，流程是包含业务流程定义的顶级 BPEL 元素。BPEL 使用<variable>声明保存每一个流程运行的上下文信息的变量。为了描述两个服务间的关系，BPEL 使用<partnerLink>定义合作伙伴链接。合作伙伴链接类型定义了关系中每个服务所扮演的角色(Role)，并指定每个角色所提供的类型端口。一个 BPEL 流程由多个步骤组成，每个步骤称作活动（Activities）。BPEL 支持两种活动：基元活动和结构活动，其中基元活动表示基本构造，用于常见任务：使用<invoke>调用其他 Web 服务，使用<receive>接收请求、等待客户端通过发送消息调用业务流程，使用<reply>生成同步操作的响应，使用<assign>操作数据变量，使用<throw>指示故障和异常，使用<wait>等待一段时间，使用<terminate>终止整个流程。然后，可以组合这些基元活动以及其他基元活动，以定义准确指定业务流程步骤的复杂算法。同时，为能够组合基元活动，BPEL 支持几个结构活动。其中最重要的是：顺序(<sequence>)允许定义一组将按顺序调用的活动；流(<flow>)用于定义一组将并行调用的活动；switch-case 构造（<switch>）用于实现分支；<while> 用于定义循环；使用<pick>能够选择多个替换路径之一。

2. 工件流技术

BPEL 所组成的流程（Process）是产生某一结果的一系列作业。流程是多个人员、多个作业按照一定的规则的有序组合。它关心的是谁做了什么事，产生了什么结果，传递了什么信息给谁。流程一定是体现企业价值的，没有价值的流程是没有意义的，因此每个流程都有其特定的目标。在信息系统中，流程由若干作业（Operation）按照一定的规则组合而成，可以用业务流程图来描述，其目标通过绩效指标体现。作业是为了实现一个可定义的目标而进行的一系列活动，是业务流程的基本单元。在信息系统中，作业的前端表现为若干界面，后端由若干个服务按照一定的规则组合而成。而关于流程最早是以 WfMC 为代表的“业务流程开发商”，工作流管理联盟（WfMC）于 1993 年成立。WfMC 主要拥护以 XPDL 作为描述语言来描述业务流程；之后是以 OASIS（Organization for the Advancement of Structured Information Standards，结构化信息标准促进组织）组织为代表的，被 IBM，MicroSoft，BEA 所拥护的 BPEL/BPEL4WS 规范；之后向来以规范著称的 OMG 组织也不甘示弱，联合 BPMI

组织，独辟蹊径以 Notation Specification 为入口，首先推出了 BPMN 规范，进而推出了 BPDM（Business Process Definition Metamodel）。于是 2003 年 4 月 BPEL 规范提交给了 OASIS 更名为 WSBPEL（Web Services Business Process Execution Language）规范。此规范描述如何处理输入的消息，它不是一个关于业务流程规格化定义的规范。简单的说，可以将它看作 XML 形式的编程语言，提供将 WSDL-Services 组合成控制流的能力。由于 BPLE 对于人工活动支持不好，为此进一步扩展为 BPEL4People（WS-BPEL Extension for People），从只能编排 Web 服务，扩展为同时支持对 Web 服务和基于角色的人工活动进行编排。

而业务流程管理（Business Process Management，BPM），一般的定义是一套达成企业各种业务环节整合的全面管理模式。BPM 涵盖了人员、设备、桌面应用系统、企业级后台应用等内容的优化组合，从而实现跨应用、跨部门、跨合作伙伴与客户的企业运作。并且根据 WfMC 的定义，工作流（Work Flow）就是自动运作的业务过程部分或整体，表现为参与者对文件、信息或任务按照规程采取行动，并令其在参与者之间传递。简单地说，工作流就是一系列相互衔接、自动进行的业务活动或任务。同时，工作流管理（Workflow Management，WFM）是人与电脑共同工作的自动化协调、控制和通信，在电脑化的业务过程上，通过在网络上运行软件，使所有命令的执行都处于受控状态。在工作流管理下，工作量可以被监督，分派工作到不同的用户达成平衡。

（1）工作流管理（WFM）组件设计。传统的办公自动化或者协同办公系统，要实现基于工作流的流转，需要有两个基本的功能：工作流引擎和自定义表单，有了这两个基本功能就可以在一个系统中实现流程的流转。但是如果要实现整合企业所有的应用（不管是什么平台、什么开发商），特别是要将所有的业务全部整合到一个工作流中，就需要工作流组件提供一个松耦合的连接方式，将所有的应用整合在一块，保证现有的系统都能最大程度的整合到统一的工作流中，从而实现统一企业的工作流。而对于作流通信模式，工作流组件和其他业务组件通过 Web 服务方式调用，并可以采用异步和同步传输模式进行传送。其中，同步传输主要适用于两个系统必须同时提交的业务场景，采用同步传输需要等待另外一个系统的反馈信息，对提交 Web 服务的系统有性能的影响；异步传输的方式主要适用于可以异步提交 Web 服务的业务场景，保证了提交 Web 服务系统的性能，异步提交的方式需要考虑接受服务的系统出现死机或者网络故障的情况下还可以准确无误的将 Web 服务传递到接收系统。异步传输一般通过消息中间件完成。

同时，为了实现松耦合，业务组件和工作流组件之间不进行业务数据交互，传递的仅仅是任务信息。业务组件仅对外提供获取信息或者写入信息的 Web 服务，使得和普通的业务组件之间的 Web 服务没有什么区别，且读取信息和写入信息是标准的业务服务，这保证了为工作流使用的 Web 服务具有通用性。如果需要和工作流整合，仅仅提供一个特殊的 Web 服务“通知完成”将任务的完成状况或者任务的基本信息等传递到工作流组件，任务的基本信息主要是为了痕迹化管理，将修改的信息做一个记录，便于未来的审计。通过这种模式实现业务数据和流程数据分离。工作流组件和业务组件不进行业务数据的交互，简化了工作流整合的难度。这时，工作流组件则提供启动流程和修改流程状态，启动下一环节和保存任务基本信息等 Web 服务。这时，为了使流程平台具有良好的扩展性，如果工作流组件需要业务数据（比如需要根据业务数据进行判断业务流转），则这时需要对进行审批的内容进行保存，并通过 BPLE 调用查询服务将数据保存到流程数据库中，其调用跟工作流引擎没有关系。实现跨系

统的流程流转，也是通过BPEL的编排，通过调用业务Web服务实现。

作为公共组件的工作流模块主要包含三个主要功能：工作流引擎、待办任务管理和工作流可视化管理。其中，工作流引擎是基础模块，可以为所有的系统提供工作流引擎，实现工作流流转的逻辑控制。待办任务管理主要实现人机交互，提供一个统一管理的待办任务管理，整合公共的工作流引擎，以及企业已经存在的工作流引擎（如OA），形成一个统一的待办任务管理。工作流可视化管理主要用于工作流的可视化展示和工作流定义，还可以实现流程监控、流程业务监控、流程导航等功能。同时，当工作流与其他业务组件集成整合实现不同的业务功能时，需要工作流管理组件和业务组件整合形成有效的紧密程度来描述，这种紧密程度可分为以下三种：① 完全独立的公共工作流管理组件，使工作流引擎和业务完全隔离开，流程任务通过Web服务方式交互；② 内嵌的工作流管理模块，即业务组件通过API的方式调用内嵌在业务组件中的工作流引擎，使得工作流引擎和公共的工作流引擎共享数据库；③ 公共工作流管理组件和内嵌工作流模块组合，即工作流引擎有自己的流程数据库，包含已有系统的工作流引擎和已有的流程数据库，使得工作流引擎和业务组件紧密集成，并通过ESB以Web服务方式或者通过数据总线和公共工作流引擎进行数据交换。内嵌的工作流引擎负责任务的处理，公共的工作流组件集成待办任务，如图3-42所示。

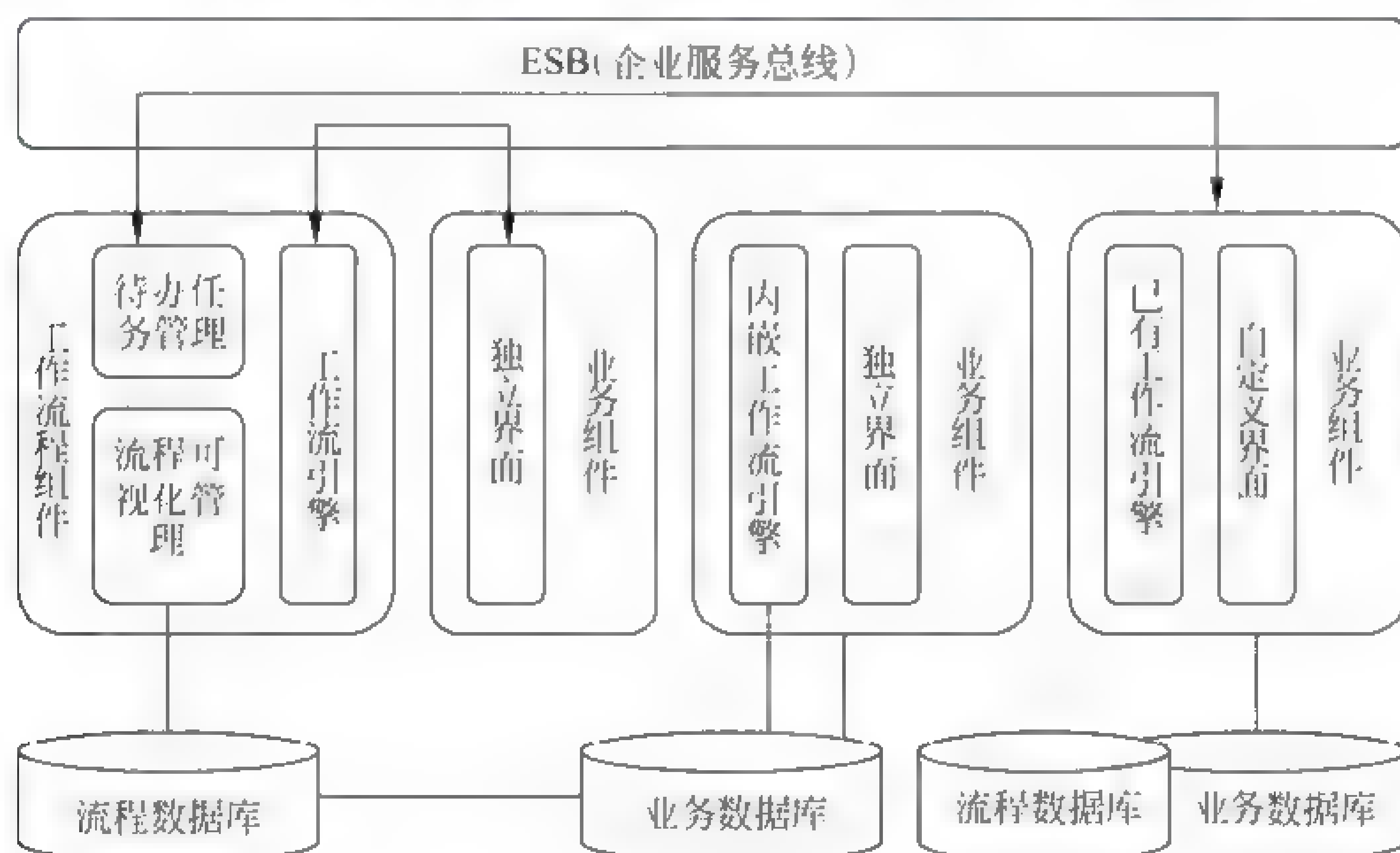

图3-42　工作流模块内容及和业务组件关系

（2）基于工作的报销审批设计模式。工作流组件在应用方面主要包含三个部分：工作流引擎、待办任务管理、工作流可视化管理。在数据库层面，主要包含两个部分：流程数据和业务操作记录，其中，流程数据保存企业所有流程的流程定义、流程运行状态和待办任务，以及流程可视化等流程本身的数据、业务数据保存流程过程中的业务数据。由于业务数据结构多样，为了保证对流程操作过程的详细记录，这时需要将所有通过工作流的业务操作信息全部直接以XML方式存放到流程数据库，如图3-43所示。下面以报销流程为例说明工作流模块的使用。

在财务系统中，报销单据凭证处理都是完整的，但是财务系统一般不支持所有员工在财务系统中录入报销单据数据（如发票号码、金额等），同时由于财务软件是产品化的软件，很难进行大的改造。因此实现每个员工填写报销单据，并由上级领导审批报销单据，最后由财

务进行审核，需要一个报销管理。报销管理单据的维护、审批处理，信息简单，可以直接用表单定制和工作流管理模块定制出来，包括填写报销单据、直接上级审批、领导审批等流程节点，就像一般的办公自动化系统实现的功能。当领导审批完成之后，启动一个业务流程（采用 BPEL 进行编排），将单据数据自动写入财务系统（只需要财务系统提供写入单据 Web 服务）。正如上面所述一样，由于财务系统无法进行大的改造，在财务审批、财务入账环节，采用消息流程结点处理方式，由财务系统提供一个查询单据状态的接口，这样财务人员就可以在待办任务中看到报销单据的待办。同时，财务人员通过单点登录，点击待办任务进入财务系统进行操作，这样就使得报销单据的填报者则不需要进入财务系统，就可以在报销系统中看到报销单据的整个流程的状态，这样既可以实现整合所有流程，又对财务系统本身不会造成大的影响，如图 3-44 所示。

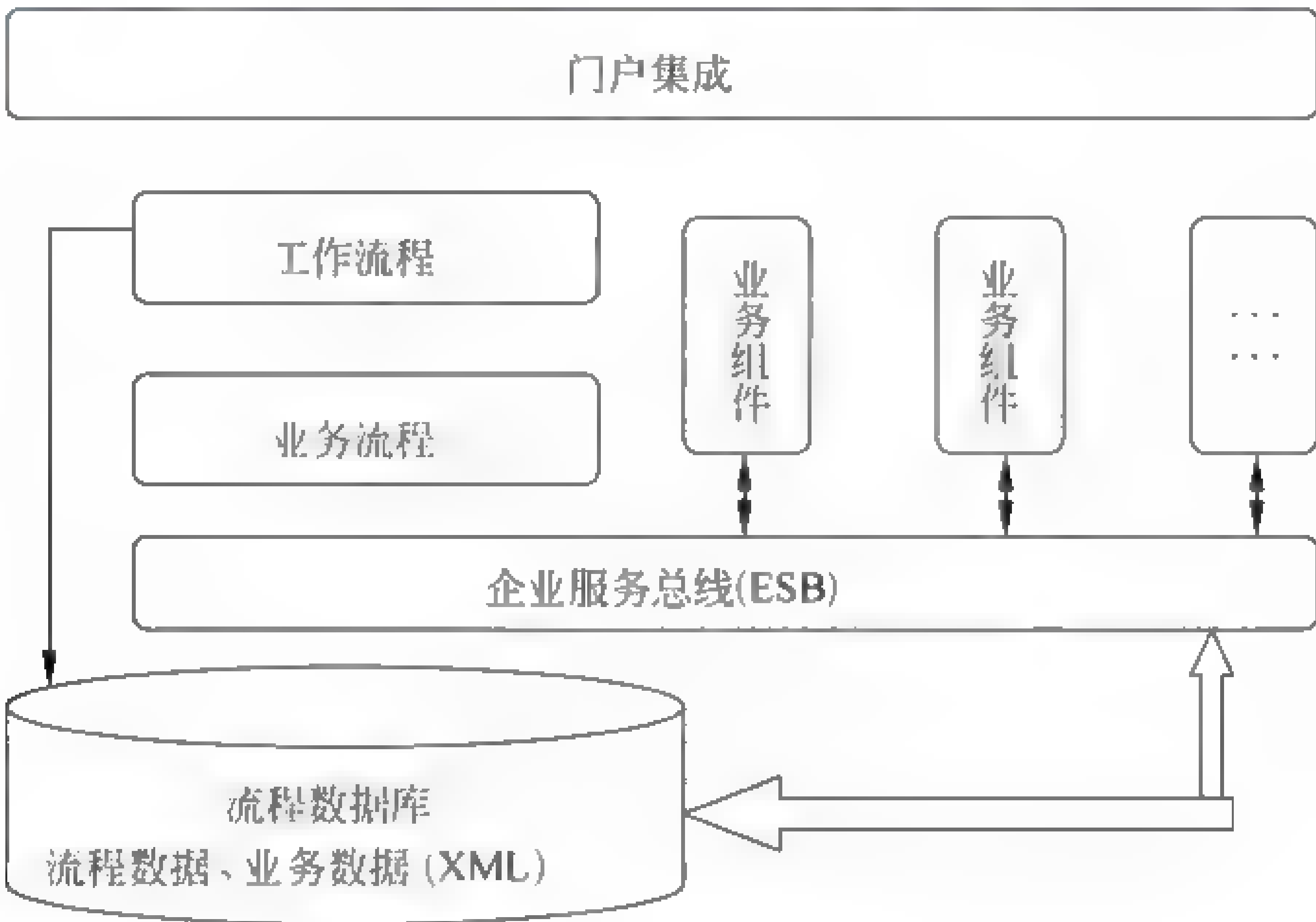

图 3-43 流程整合

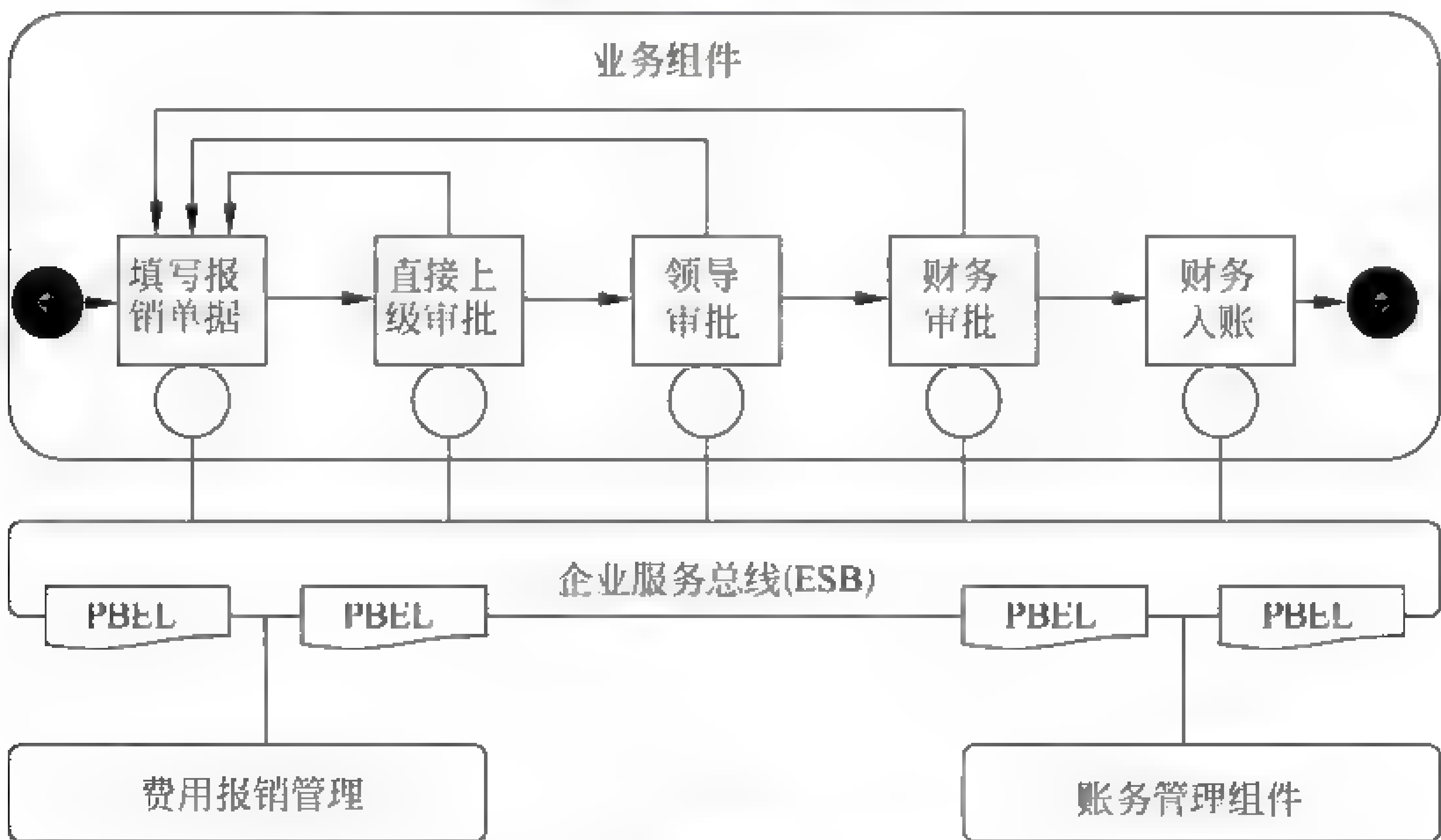

图 3-44 报销流程举例

3.3.3.4 基于 SOA 的 UML 与 BPEL 转换方法

近年来随着对象管理组织（OMG）对模型驱动架构（MDA）的推广，使得 MDA 在软件

开发过程中起着越来越重要的位置。而如何使得 MDA 在 Web 服务世界中更好地应用成为了一项重要的研究课题。国内外一些研究机构和 IT 企业在这方面作了不同程度的研究，例如将 Web 服务模型转换成多种不同的实现平台：Java、.Net、Delphi 等。然而这些研究都仅仅集中于 Web 服务模型的静态结构转换。基于此，这时需要将研究 Web 服务模型的动态行为转换，这些动态行为模型描述了多种相互协作的完成一定任务或者实现系统一定功能的组件。元模型在整个模型驱动架构中扮演着重要的角色，它提供了用来描述模型的元素、规则。通常用 UML 活动图的元模型来表示 BPEL 结构，其中包含了代表工作流（Workflow）、对象流（Object Flow）、活动（Activities）、操作（Actions）以及分叉（Fork）、联结（Join）等表示控制结点的元模型元素。这些元模型元素来源于 UML2.0 Superstructure Specification 中各种活动、操作模型元素。

基于 MDA 的模式是对 UML 一种支持，为在实现 UML 与 BEPL 转换时提供了技术支撑。正在前面所述，UML 是一种 OMG 标准，该标准提供了一种可视化的建模表示法，这对设计和了解复杂的系统很有效。这是由于 UML 具有以下优点：

（1）是众所周知的面向对象（OO）建模表示法；

（2）具有非常容易理解的图形表示法和一套丰富的语义集来捕获 OO 系统的关键特征；

（3）UML 广泛地应用于面向对象的软件开发，还常用于定制的、基于组件的软件开发、业务流程建模和系统设计。

扩展或定制 UML 的特性对 MDA 来说是很必要的；可以通过定制 UML 来支持系统建模，这种系统是需要完全或部分的部署到 Web 服务基础架构上，并且通常模板（Stereotypes）是一种对模型的元素进行分类的方法。而 BPEL 可以看作是能够编排和协调 Web 服务的描述语言 WSDL 的一种扩展。也就是说 WSDL 是用来描述独立的 Web 服务及其结构信息的，而 BPEL 描述的则是一系列 Web 服务的编排和协调的方式。因此，可以用 UML 类图描述 Web 服务的静态结构来表达 WSDL 的语义。当前已经有一些研究机构在关于 UML 类图与 WSDL 之间的映射关系和模型转换方面作了实际的工作。下面叙述 Web 服务中动态模型与 UML 活动图之间的映射关系及模型转换的方法。

模型转换定义了如何从一种源模型转换成另外一种目标模型。模型转换的研究也有多年了，但直到目前为止也还没有一个统一的标准。现在对象管理组织（OMG）在 2008 年 4 月发布了为 QVT（Query，View，Transformation）的项目，它是属于 MOF 范畴，并且基于 QVT 的开源软件也被 Sourceforge 接纳，目前的版本是 0.98（目前可以从 http://umt-qvt.sourceforge.net/downloads.html 处下载），它的目的是为变换工具制定一个标准，如图 3-45 所示。一般情况下，模型转换是通过 PIM（平台无关模型）和 PSM（平台相关模型）模型来实现的。

PIM 模型使用平台无关的语言来说明，这种平台无关的语言使用平台无关的元模型来描述。同样的，PSM 模型使用平台相关的语言来说明，这种平台相关的语言同样使用平台相关的元模型来描述。为实现从 PIM 生成 PSM 的转换目的，须存在一个转换规则，按照此规则将平台无关的元模型转换为平台相关的元模型。本书中平台无关的元模型是 UML2.0 活动图元模型，平台相关的元模型是 BPEL1.1 的元模型，转换规则是基于 OCL 的规则约束，转换过程由一个转换引擎来完成。

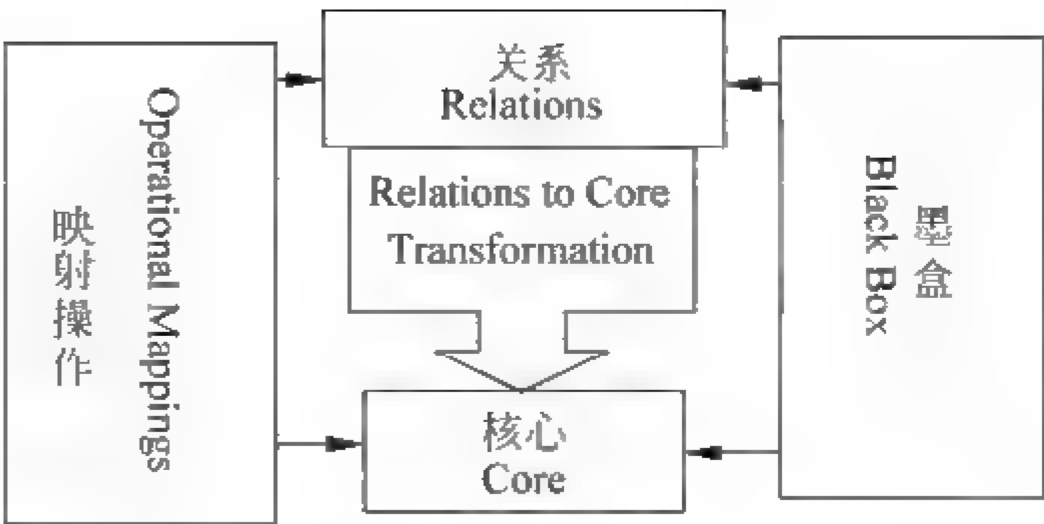

图 3-45 QVT 元模型间的关系

从 UML 活动图元模型到 BPEL 元模型的映射表示可以从 UML 活动图模型生成的 BPEL 模型。表 3-9 概要地展示了从 UML 2.0 活动图元模型元素到 BPEL 元模型元素的映射，覆盖到了本书介绍的模型元素。

表 3-9 UML 2.0 活动图元模型元素到 BPEL 元模型元素的映射

UML 2.0 活动图元模型元素	UML 2.0 描述	BPEL 标签	BPEL 描述
Activity	一种基本的行为方式，本质上指可包含一系列动作和相关变量的行为序列	process	BPEL 流程定义
ObjectNode, Structured-ActivityNode (Variable), DataStore	对象结点 ObjectNode 用来指定对象数据流中的值；结构活动结点 Structured Activity Node 是一种特殊的可执行结点，可包含其他活动结点及变量；数据存储 DataStore 用来存放缓冲数据信息	variable	保存每一个流程运行的上下文信息
ControlFlow	控制流 ControlFlow 包含一个或多个按照一定顺序执行的活动	sequence	定义一组活动的调用顺序
AcceptCallAction	用来接收请求信息的动作，以响应客户端的请求；等待客户端发送请求信息以启动活动流	receive	使用<receive>接收请求、等待客户端通过发送消息调用业务流程
CallAction, CallOperationAction	操作调用，将请求动作转发给目标对象；可以是同步的，也可以是异步的	invoke	流程用<invoke>活动调用伙伴提供的 Web 服务；可以是同步的，也可以是异步的
VariableAction, ReadVariableAction, WriteVariableAction	对变量进行读写操作，可以将一个变量赋予另一变量	assign	用于将数据从一个容器复制到另一个容器，也可以用于通过使用表达式构造和插入新数据
ReplyAction	接收一系列返回值，并用来响应 AcceptCallAction 动作	reply	在流程中可以定义多个 reply 活动来回答该伙伴的调用

同时，自动化业务流程的 UML 配置文件表示从 UML 模型生成完整的可执行 BPEL 制品。表 3-10 所示概要的展示了从配置文件到 BPEL 的映射，覆盖到了本书介绍的配置文件子集。

表 3-10　BPEL 的映射关系

配置文件构件	BPEL4WS 概念
<<process>>类	BPEL 流程定义
<<process>>类的活动图	BPEL 活动级别
<<process>>类属性	BPEL 变量
分层结构和控制流	BPEL 顺序和流程活动
<<receive>>、<<reply>>、<<invoke>>活动	BPEL 活动

下面以一个简易的贷款审批流程为例来说明 UML 与 BPEL 转换的基本方法：在收到贷款请求时，将请求的数值与数值（贷款最高限额是 10000）比较。如果请求的数值比较少，那么将调用 Assessor 服务，否则将将调用 Approver 服务。如果 Accessor 认为该请求的风险比较高，它也将被传递给 Approver。当 Approver 完成或者 Accessor 接受时，将会返回批准信息。BPEL 流程是有状态的并包含实例，所以在 BPEL 中，这种情况都会被作为一个 LoanApproval 流程而实现，对于被处理的每个实际贷款申请，LoanApproval 流程都会有一个实例。每个实例都用 BPEL 变量来捕获它自己的状态。在 UML 配置文件中，流程被表示为 <<Process>> 模板类，流程代码如下：

```
<process name="loanApprovalProcess" ...>
  <variables>
    <variable name="request"
              messageType="loandef:creditInformationMessage"/>
    <variable name="riskAssessment"
              messageType="asns:riskAssessmentMessage"/>
    ...
  </variables>
  ...
   <flow>
      <receive name="receive1" partner="customer"
            portType="apns:loanApprovalPT" operation="approve" variable=
            "request"
            createInstance="yes">
      <source linkName="receive-to-assess"
             transitionCondition=
             "bpws:getVariableData('request', 'amount')<10000"/>
      <source linkName="receive-to-approval"
             transitionCondition=
             "bpws:getVariableData('request', 'amount')>=10000"/>
    </receive>
    <invoke name="invokeAssessor" partner="assessor"
           portType="asns:riskAssessmentPT"
           operation="check"
           inputVariable="request"
           outputVariable="riskAssessment">
      <target linkName="receive-to-assess"/>
      <source linkName="assess-to-setMessage"
             transitionCondition=
```

```
            "bpws:getVariableData('riskAssessment', 'risk')='low'"/>
      <source linkName="assess-to-approval"
            transitionCondition=
            "bpws:getVariableData('riskAssessment', 'risk')!='low'"/>
    </invoke>
    <assign name="assign">
      <target linkName="assess-to-setMessage"/>
      <source linkName="setMessage-to-reply"/>
      <copy>
        <from expression="'yes'"/>
        <to variable="approvalInfo" part="accept"/>
      </copy>
    </assign>
    ...
    <reply name="reply" partner="customer" portType="apns:loanApprovalPT"
          operation="approve" variable="approvalInfo">
      <target linkName="setMessage-to-reply"/>
      <target linkName="approval-to-reply"/>
    </reply>
  </flow>
</process>
```

BPEL 可以通过活动图来清楚的描述类的行为。图 3-46 显示了贷款批准流程的活动图。例如，invokeAssessor 活动显示为带有圆角的长方形。执行的操作显示为活动的入口条件；例如，riskAssessment（一个变量）被设置为检查服务的结果。通过 UML 分区（也作为泳道）来表示流程中通信的合作伙伴：customer、assessor 和 approver。每个分区中显示了往合作伙伴发送或者接收信息的活动。箭头表示流程执行活动的顺序。注意分派（assingment）活动没有放到一个分区中；它描绘了发生在它自身流程内的活动，该流程不需要外部服务。

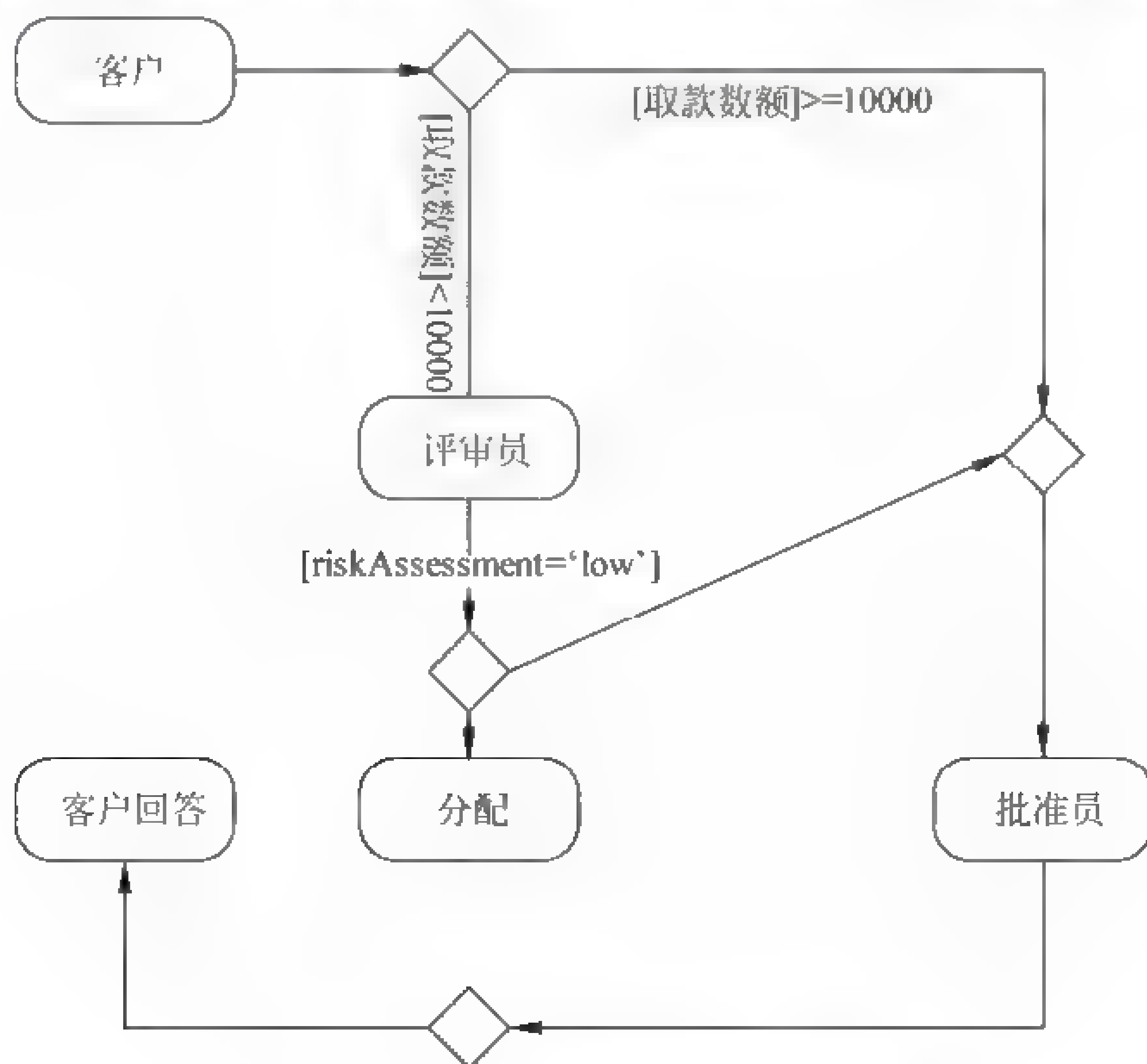

图 3-46 一个简易贷款审核流程

3.3.4　面向企业服务总线的方法

ESB 是一种在松散耦合的服务和应用之间标准的集成方式。它可以作用于：

（1）面向服务的架构——分布式的应用由可重用的服务组成；

（2）面向消息的架构——应用之间通过 ESB 发送和接受消息；

（3）事件驱动的架构——应用之间异步地产生和接收消息。

ESB 就是在 SOA 架构中实现服务间智能化集成与管理的中介，如图 3-47 所示。而它与 SOA 的关系要相对好理解一些，ESB 是逻辑上与 SOA 所遵循的基本原则保持一致的服务集成基础架构，它提供了服务管理的方法和在分布式异构环境中进行服务交互的功能。可以这样说，ESB 是特定环境下（SOA 架构中）实施 EAI（Enterprise Application Integration）的方式。

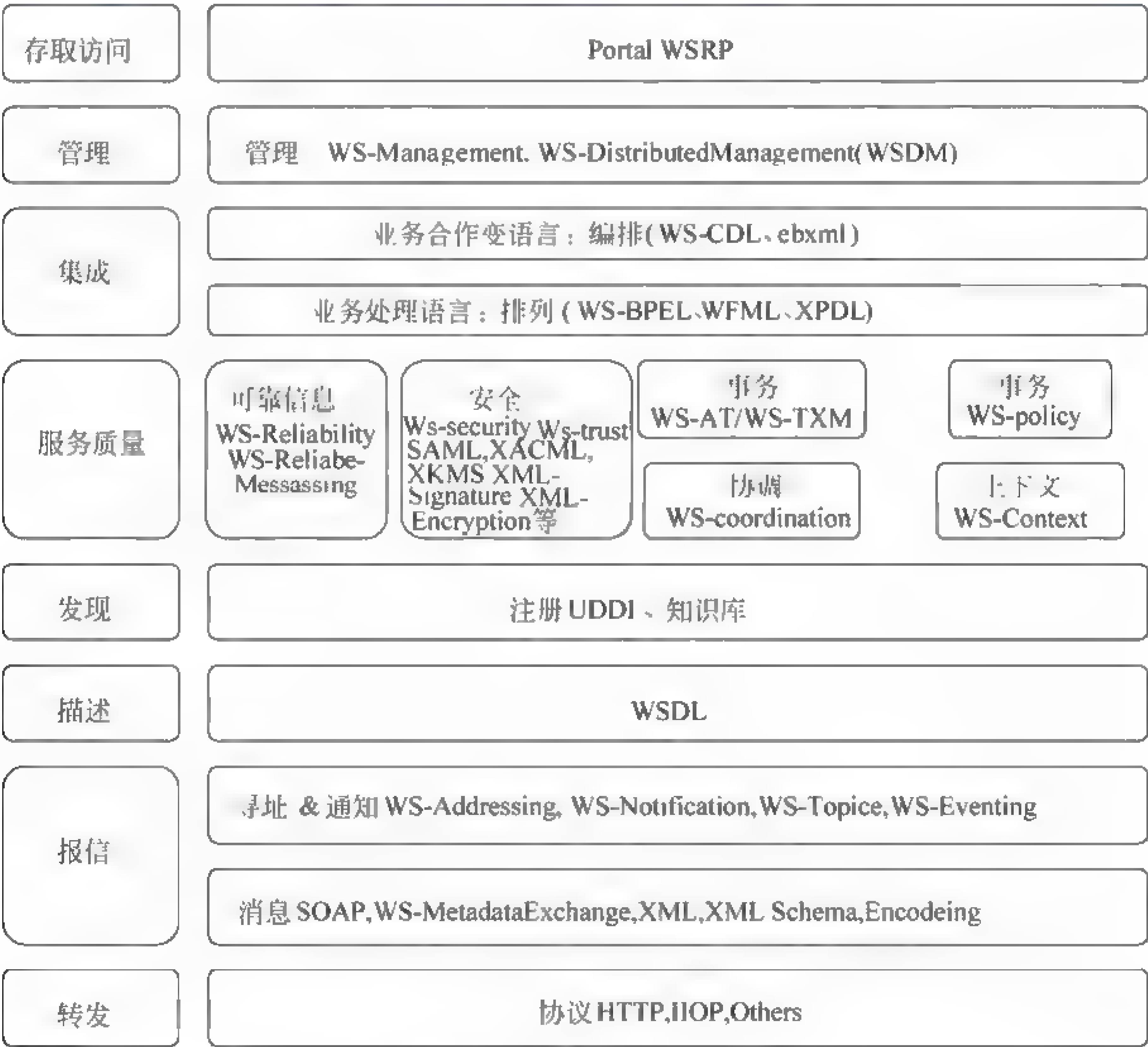

图 3-47　ESB 体系结构

（1）在企业服务总线（ESB）系统中，被集成的对象被明确定义为服务，而不是传统 EAI 中各种各样的中间件平台，这样就极大简化了在集成异构性上的考虑，因为不管有怎样的应用底层实现，只要是 SOA 架构中的服务，它就一定是基于标准的。

（2）ESB 明确强调消息（Message）处理在集成过程中的作用，这里的消息指的是应用环境中被集成对象之间的沟通。以往传统的 EAI 实施中碰到的最大的问题就是被集成者都有自己的方言，即各自的消息格式。作为基础架构的 EAI 系统，必须能够对系统范畴内的任何

一种消息进行解析。而传统的 EAI 系统中的消息处理大多是被动的，消息的处理需要各自中间件的私有方式支持，例如 API 的方式。

（3）事件驱动成为 ESB 的重要特征。通常服务之间传递的消息有两种形式，一种是调用（Call），即请求/回应方式，这是常见的同步模式。还有一种称为单路消息（One-Way），它的目的往往是触发异步的事件，发送者不需要马上得到回复。考虑到有些应用服务是长时间运行的，因此，这种异步服务之间的消息交互也是 ESB 必须支持的。

企业服务总线（ESB）也是一种体系结构模式，可以采用许多不同的产品在组织内实现，并组装起来作为联合总线。它将事件驱动体系结构的方法（Event-Driven Architecture，EDA）和面向服务的方法结合使用，以简化业务单元的集成，从而在异类平台和环境间建立联系。ESB 充当允许不同应用程序进程之间进行通信的中间层，从而部署到企业服务总线的服务可以由使用者或事件触发。它同时支持同步方式和异步方式，可促进一个或多个参与者之间的交互（一对一和多对多通信）。表 3-11 和表 3-12 所示是 ESB 可提供 SOA 和 EDA 范例的所有功能。

表 3-11　ESB 在可提供的基本 SOA 特征

功能	描述
松散耦合的交互	服务的调用独立于其技术和位置
一对一通信	一个特定服务一次由一个用户调用。通信是双向的
基于用户进行触发	控制流由客户机（服务使用者）发起
同步	应答将以同步方式发回给使用者

表 3-12　ESB 可提供的基本 EDA 特征

功能	描述
分离的交互	事件发布者并不会意识到事件订阅者的存在
多对多通信	采用发布/订阅消息传递，一个特定事件可以影响多个订阅者
基于事件的触发器	控制流由接收者确定（基于发布的事件）
异步	通过事件消息传递支持异步操作

3.3.4.1　ESB 功能描述

ESB 支持异构环境中的服务、消息，以及基于事件的交互，并且具有适当的服务级别和可管理性。为了达到此目的，需要将多种功能集中起来并加以分类。使其 ESB 描述为由中间件技术实现并支持 SOA 的一组基础架构功能。ESB 功能包括如表 3-13 所示：

表 3-13　ESB 的功能

所支持功能名称	所具有的功能描述
通信	具有路由、寻址，通信技术、协议和标准（HTTP 和 HTTPS），发布/订阅，响应/请求，Fire-and-Forget、事件，同步和异步消息传递
集成	数据库、服务聚合、遗留系统和应用程序适配器、EAI 中间件的连接性、服务映射、协议转换、应用程序服务器环境（例如 J2EE 和.NET）、服务调用的语言接口（例如 Java 和 C/C++/C#）
安全性	身份验证、授权、不可抵赖性、机密性、安全标准[例如 Kerberos 和 Web 服务安全性（WS-Security）]

续表

所支持功能名称	所具有的功能描述
消息处理	编码的逻辑、基于内容的逻辑、消息和数据转换、有效性、中介、对象标识映射、数据压缩
建模	对象建模、通用业务对象建模、数据格式库、B2B 集成的公共与私有模型、开发和部署工具
服务交互	服务接口定义（例如，Web 服务描述语言（Web Services Description Language，WSDL）、支持替代服务实现、通信和集成所需的服务消息传递模型[例如 SOAP 或企业应用程序集成（EAI）中间件模型] 服务目录和发现
服务质量	事务[原子事务、补偿、Web 服务事务（WS-Transaction）]、各种确定的传递范例[例如 Web 服务可靠消息传递（WS-ReliableMessaging）或对 EAI 中间件的支持]
服务级别	性能、吞吐量、可用性、其他可以构成契约或协定的持久评估方法
管理和自治	服务预置和注册，记录、测量和监控，发现，系统管理和管理工具的集成，自监控和自管理
基础架构智能	业务规则；策略驱动的行为，特别是对于服务级别、服务功能的安全和质量[例如 Web 服务策略（WS-Policy）]；模式识别

在实现基于 SOA 的 ESB 时，首先启用 SOA 应用程序涉及到创建服务接口，这是因为服务接口可以直接或间接地通过使用适配器来集成现有的或新的功能。从最基本的级别来看，启用该基础架构涉及到规划功能来将服务请求路由和传递给正确的服务提供者。然而，基础架构支持在不影响服务的客户端的情况下由另一个服务实现替代原有的服务实现也是至关重要的。这不仅需要根据 SOA 原则指定服务接口，而且需要基础架构允许客户端代码以独立于所涉及的服务位置和通信协议的方式来调用服务。这样的服务路由和替代是 ESB 的许多功能中的一部分。这些功能包括：

（1）利用显式的与实现无关的接口来定义服务。

（2）利用强调位置透明性和可互操作性的通信协议。

（3）封装可重用业务功能的服务的定义。

并且 ESB 是一种逻辑体系结构组件，它提供与 SOA 的原则保持一致的集成基础架构。同时，SOA 原则需要使用与实现无关的的接口、强调位置透明性和可互操作性的通信协议、相对粗粒度和封装可重用功能的服务定义。当然，采用 ESB 也可以将协同化的组件、业务功能集成在一起实现高效数据通信，提高协同通信效率和异步、同步操作。

3.3.4.2　ESB 结构描述

通常 ESB 以中介的身份出现，就必须有两方面的考虑，

（1）它必须了解被它中介的两个端点：

① 服务的请求者，以及请求者对服务的要求；

② 服务的提供者和它所提供服务的描述。

（2）它必须具有某种机制能够完成中介的任务。

把这两类考虑归纳为 ESB 的两个基本功能：面向服务的原数据（MetaData）管理功能和中介（Mediation）功能。作为 SOA 的重要构成部分，ESB 承担的重任还包括怎样将企业架构中已存在的业务服务连接到总线上来，则称为适配器（Adapter）功能。尽管服务本身已经用公开的接口来描述，但具体的实现还是运行在不同的环境中，因此，ESB 还应该提供对服

务底层协议的支持，譬如应用协议 J2ee，.NET，通信协议如 Http，JMS 等等。除此之外，还需要对具体应用中涉及到的服务加以管理，如性能，可靠性，安全性等。

ESB 提供了最基本的功能来保障 SOA 系统的运行，这些功能应该包含下列内容：

（1）在总线范畴内对服务的注册命名及寻址管理功能。

（2）面向服务的中介功能：提供位置透明性的服务路由和定位服务。

（3）多种消息传递型式（请求/响应，单路请求，发布/订阅等等）和支持广泛使用的传输协议（Http，JMS，MQ 等等）。

（4）支持多种服务集成方式，比如 JCA、Web 服务、Messaging、Adaptor。

（5）对服务管理的支持，如服务调用的记录、测量和监控数据的提供。

这些功能通常采用协议转换模型（用于当服务的请求者与服务提供者基于不同协议时的消息转换情形）、消息广播模式（用于事件驱动多个动作或者消息广播的情形）、服务匹配模式（用于需要动态选择服务提供者的情形，例如可以根据消息的内容，或负载情况，或服务级别约定（SLA），来为服务请求者选择合适的服务）和服务需求驱动模式（就是根据服务需求来实现 ESB 信息分布和集成）来实现与 SOA 的整合、通信和数据交换。这是因为 ESB 有以下好处：

（1）基于标准的连接。作为很多异类应用程序间的集成中枢，ESB 必须提供很多不同的集成技术，并对大量供选择的标准技术加以利用。同时，在 ESB 中的消息传递集成通常支持 JMS（Java Message Service）API，而企业信息系统的连接则是由 J2EE Connector Architecture (JCA)提供的。为了确保 Web 服务互操作性，ESB 支持 JAX-RPC 编程模型。并且不同的 ESB 组件间的集成可以通过 Java Business Integration（JBI）规范进行标准化，图 3-48 是 JBI 顶层结构图。

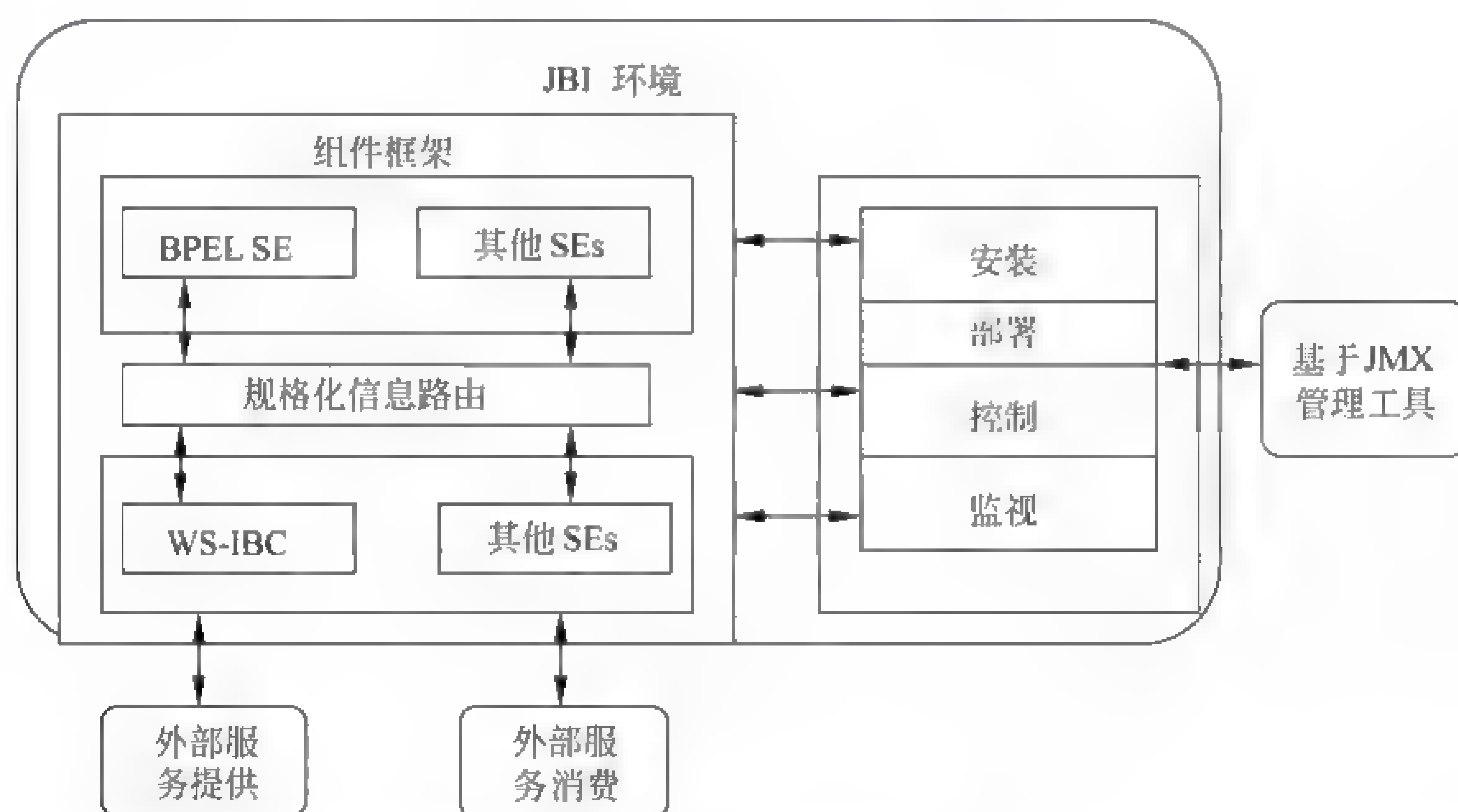

图 3-48　JBI 顶层结构图

（2）渗透性集成。ESB 具有渗透性本质，因为它可以跨不同的部门、业务单元甚至业务合作伙伴进行应用程序集成。而且，它的核心体系结构原则还可以促进构建于异类开发环境上的应用程序之间的通信。例如，ESB 解决方案可以在不同的编程语言（J2EE、C++ 或 .NET）之间起到桥梁作用。

（3）可靠集成。ESB 体系结构模式可提供系统安全性、可伸缩性或可用性。且 ESB 可以使用 SOA 和 EDA 来同时提供同步和异步功能，使传输服务可确保可靠交付和事务完整性。因此，ESB 的每个特征都对其稳健性进行了增强，可尽可能减少集成或联合解决方案失败的风险。

3.3.4.3　ESB 应用模式

ESB 是调用服务的客户机和这些服务的提供者之间的中介，它负责处理它们之间的连接任务，从而简化了客户机和提供者。通常 ESB 应用具有以下特性：

（1）通用性：提供跨整个扩展企业环境的连接层。

（2）异构性：提供面向消息的多平台、多协议和多 API 支持层，能够整合异构系统。

（3）互操作性：提供对开放协议的支持，支持来自多个供应商的系统之间的互操作。

（4）增量集成：提供按需求动态扩展系统的能力。

（5）服务质量：提供各种服务质量，如安全、性能、可靠性、可伸缩性等等。

（6）替换：使用开放 API 以确保能够替换供应商的实现。

（7）事件定位：将生成业务事件的应用程序进行定位，并提交与响应这些事件的应用程序进行分离。

（8）服务定位：通过遵循 SOA 设计方法论，使关键功能抽象为服务的方式来提供分离应用的功能等原则。

图 3-49 是一个直观的 ESB 序列图。

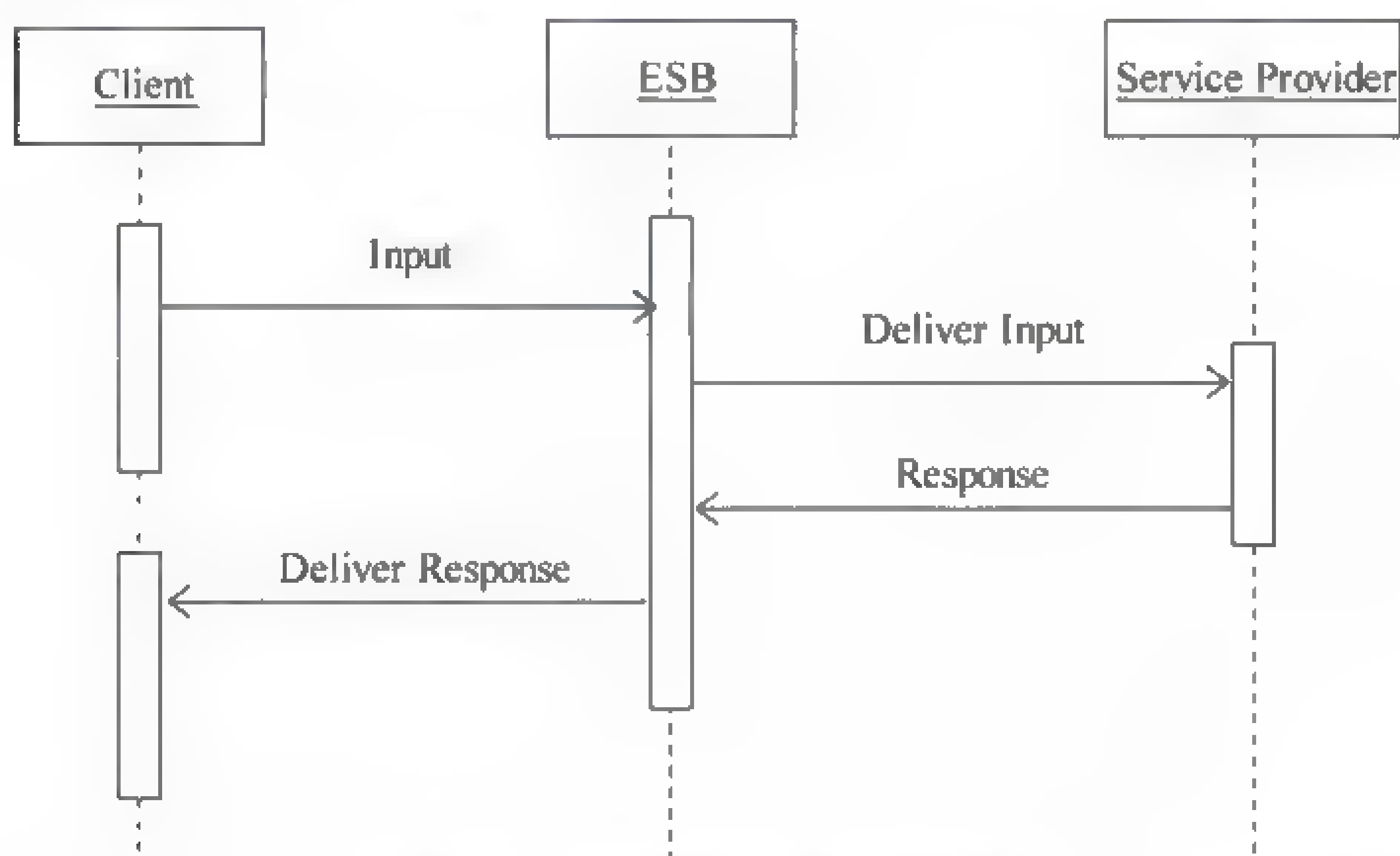

图 3-49　直观的 ESB 序列图

同样，ESB 也提供了请求与响应、数据转换、基于内容的路由、各种优化、各种服务通用连接、虚拟化和监视等功能，同时还支持 SCA 简化模型，即 ESB 使用服务注册中心和存储组件作为动态查找机制来提供服务端点的信息，如图 3-50 所示。

同时，ESB 与服务交互点（SIP）构成如下（具体可参见 3.3.2.5 小节）：

（1）使用门户（Portlet）：提供服务内容和交互数据的能力和功能；

（2）流程服务：根据业务流程和业务流，管理消息流和多个服务之间的交互的控制功能；

（3）信息服务：联合、整合、复制和转换不同数据源的功能；

（4）外部服务：将外部服务 EDI 和遗留系统集成到共同的企业体系结构中的功能；

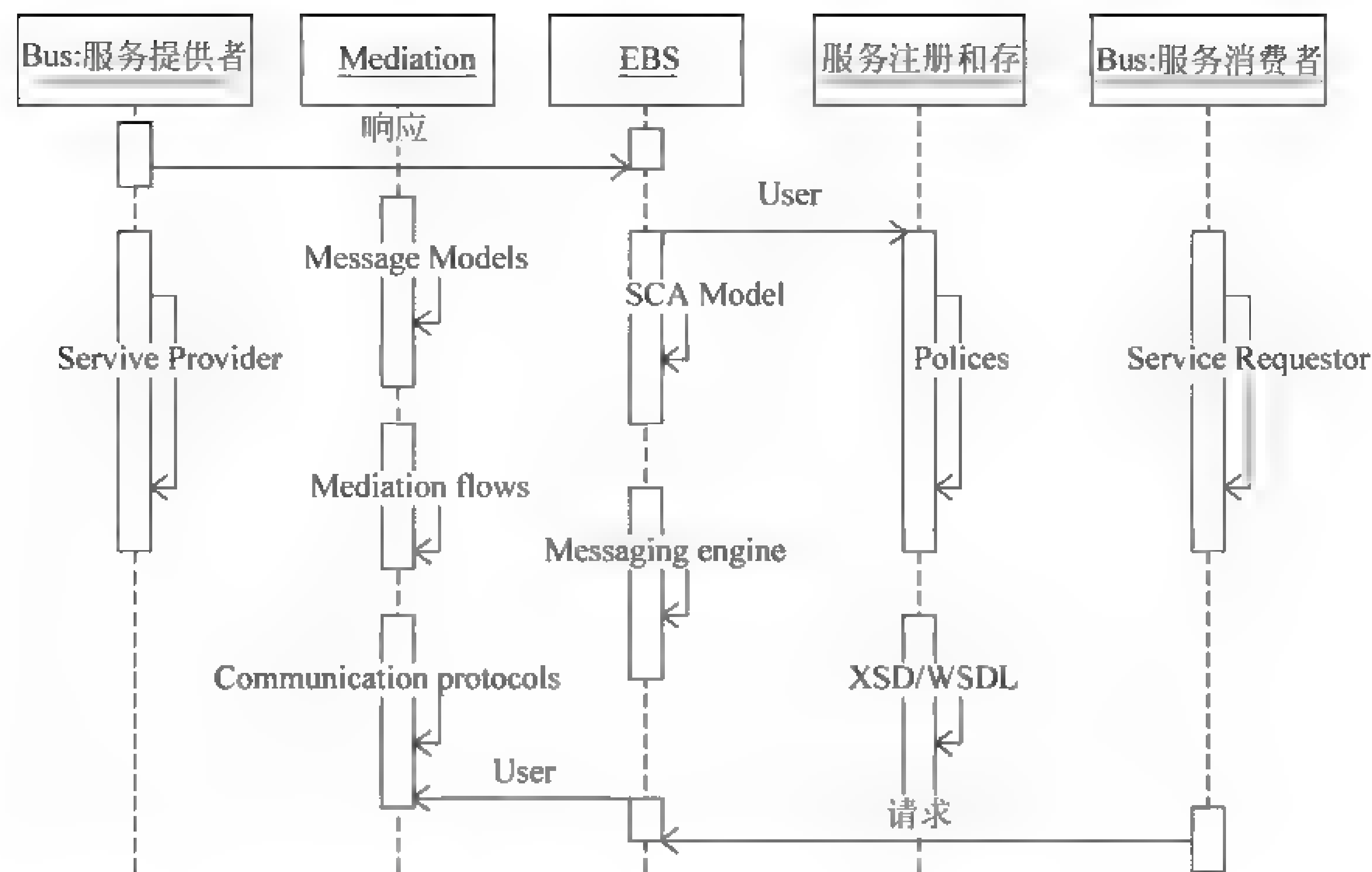

图 3-50　ESB 序列 UML 直观图

（5）本地服务：业务应用程序服务调用服务使用者的功能；

（6）应用程序和数据访问数据：将核心应用程序与外部数据存储以及打包的应用程序进行集成的功能，如图 3-51 所示。

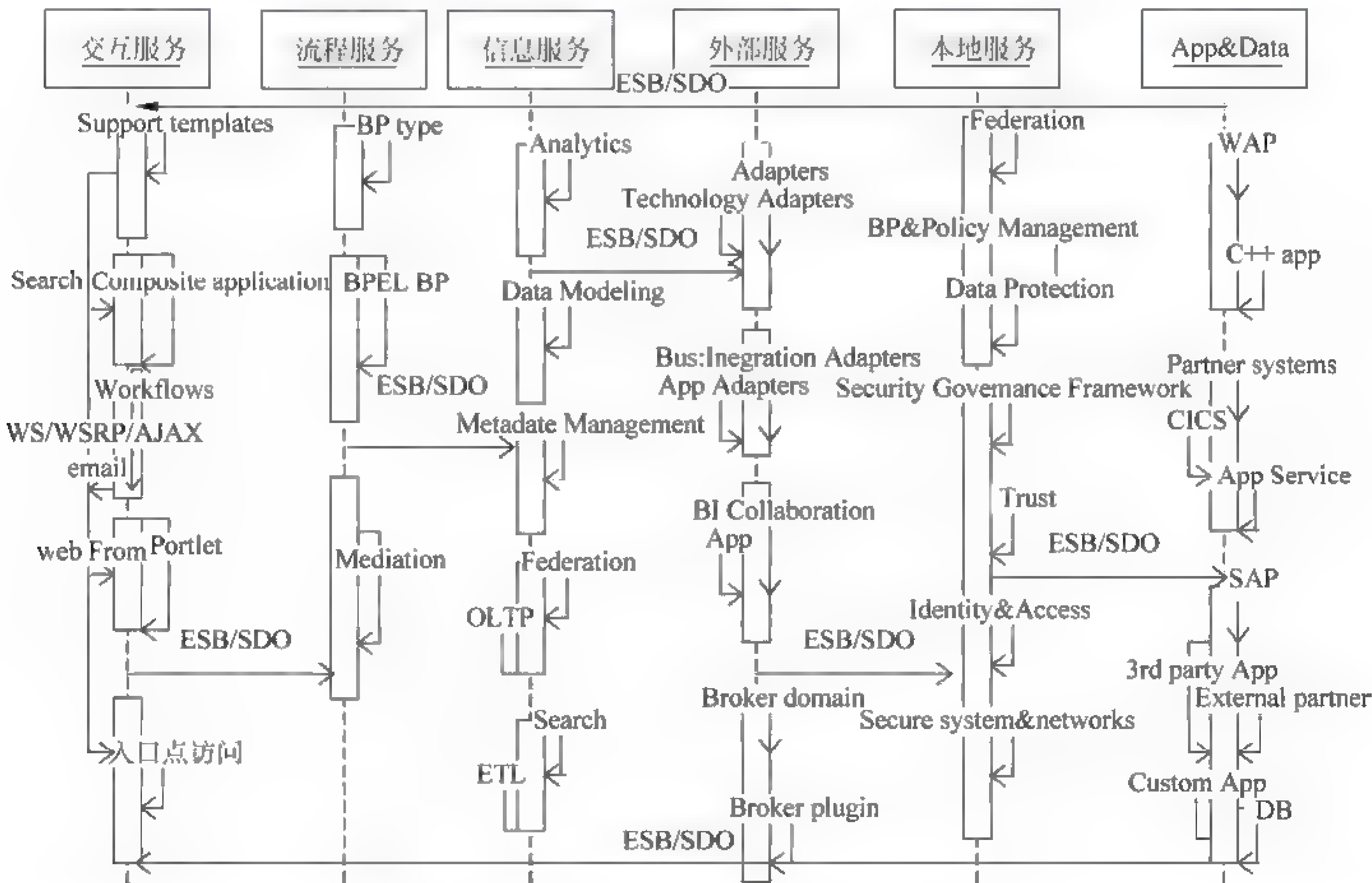

图 3-51　具有 ESB 的 SIP UML 表示图

这些服务可以是来自不同区别的协同区域和不同的业务功能，以及不同的用户，这样通过 ESB 将这些服务进行协同处理，协同交互，从而提高协同处理能力和数据利用效率。

3.3.4.4　ESB 互操作

当今存在许多与 ESB 互操作性相关的重要标准，如图 3-52 所示。下面介绍一种用于实现 ESB 互操作性的特定标准。其中包括用于消息传输（HTTP 和 HTTPS）、消息格式和协议（SOAP）以及标识或位置（WS-Addressing）的标准。在标准中将看到许多场景，并演示了如何使用基于标准的方法来满足特定的客户用例需求。一个 ESB 提供了分布的、可配置的基础设施，以及路由、转换和转化方法。其中，路由是指 ESB 在服务请求与服务响应间扮演一个匹配制造者；转换是指 ESB 操纵不同的协议或具备的交互作用；转化指 ESB 能被规划可能发生和能给予相应配合的服务接口，且提供面向方面的连接。表 3-14 所示是 ESB 在服务下所支持的标准。

图 3-52　ESB 互操作轮廓图

表 3-14　在服务下 ESB 互操作的标准

主题	标准	
	需求	特征
转移	HTTP 1.1, HTTPS	
信息格式和协议	SOAP 1.1 / 1.2	SOAP with Attachments
	MTOM/XOP	
	XML1.0	
身份或位置	WS-Addressing 1.0	
服务质量	WS-Security 1.1,	WS-SecureConversation 1.3,
	WS-ReliableMessaging 1.1	WS-Trust.
		WS-Federation
服务定义	WSDL 1.1, XML Schema	SCDL

（1）ESB 互操作路由。是指在由各类协议和安全验证支持下实现来自各方的信息格式的数据交换，以达到面向服务的业务流程互操作。如图 3-53 和表 3-15 所示。

（2）ESB 互操作转换。指在实现 ESB 互操作的路由信息交换的数据格式转换，从而实现不同的业务端的需求，如图 3-54 和表 3-16 所示。

图 3-53　ESB 互操作的路由

表 3-15　ESB 互操作中路由所需要的支持基本标准

需求	需要的规格
转移	HTTP 1.1
信息格式和协议	SOAP 1.1 or 1.2
身份或位置	WS-Addressing 1.0
服务定义	XML Schema、WSDL 1.1

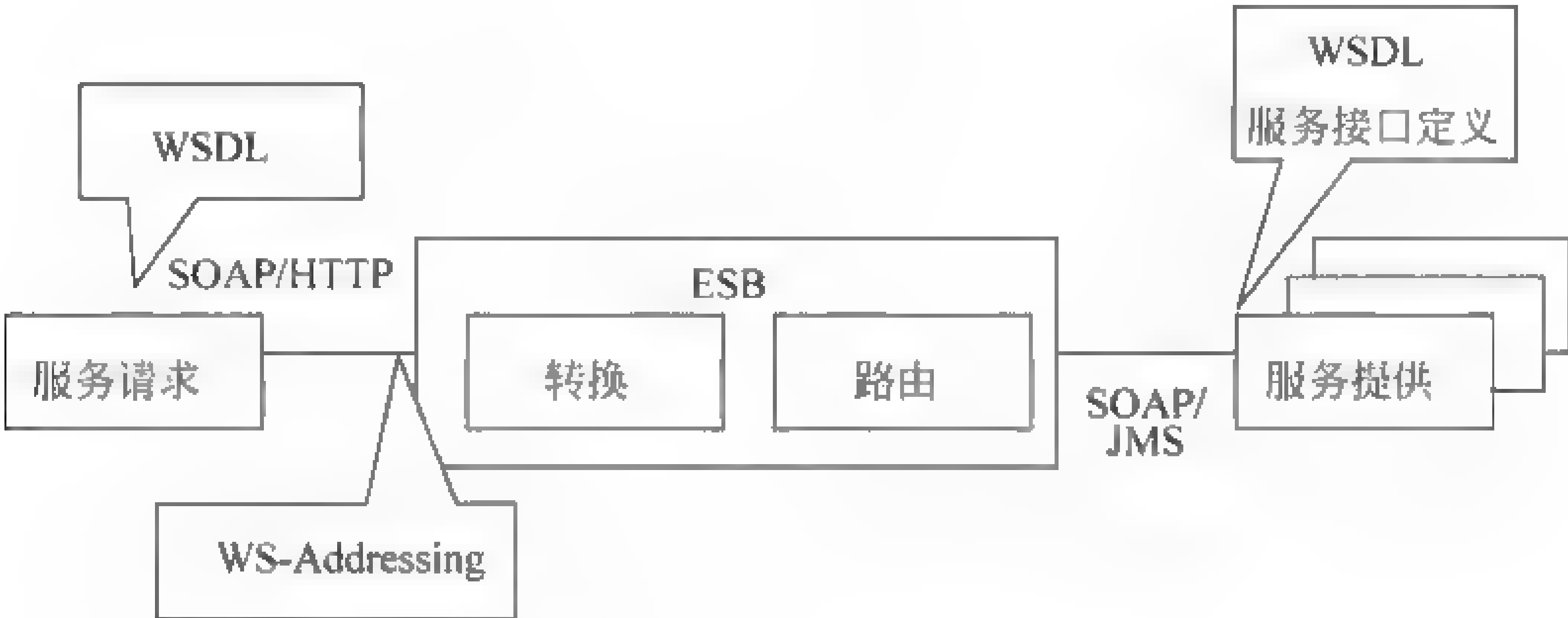

图 3-54　ESB 互操作的转换

表 3-16　ESB 互操作中转换所需要的支持基本标准

需求	需要的规格
转移	HTTP 1.1、JMS
信息格式和协议	SOAP 1.1 or 1.2
身份或位置	WS-Addressing 1.0
服务定义	XML Schema、WSDL 1.1

3.3.4.5　ESB 与 BPEL 的选择

在设计 SOA 解决方案时，并不清楚应该是使用 Web 服务的 BPEL 流程，还是使用 ESB 中介流。这是由于在现实在软件系统中，通常包含相同、相似的功能区。这就直接造成在选择不同模式、技术面临诸多问题，但只要从每一项模式、技术的概念和特征进行分析，也不难选择到合适的模式和技术。

ESB 是一种体系结构模式，而不是软件产品，也就是说不同的软件产品可以构成 ESB。而 BPEL 是 OASIS 标准组织提出的一种业务流程标准（WS-BPEL），是一种流程执行语言。WS-BPEL 定义了如何表示业务流程中的活动，以及流控制逻辑、数据、消息相关性和异常

处理等，如图 3-55 所示。ESB 与 BPEL 的主要特征区别如表 3-17 所示。

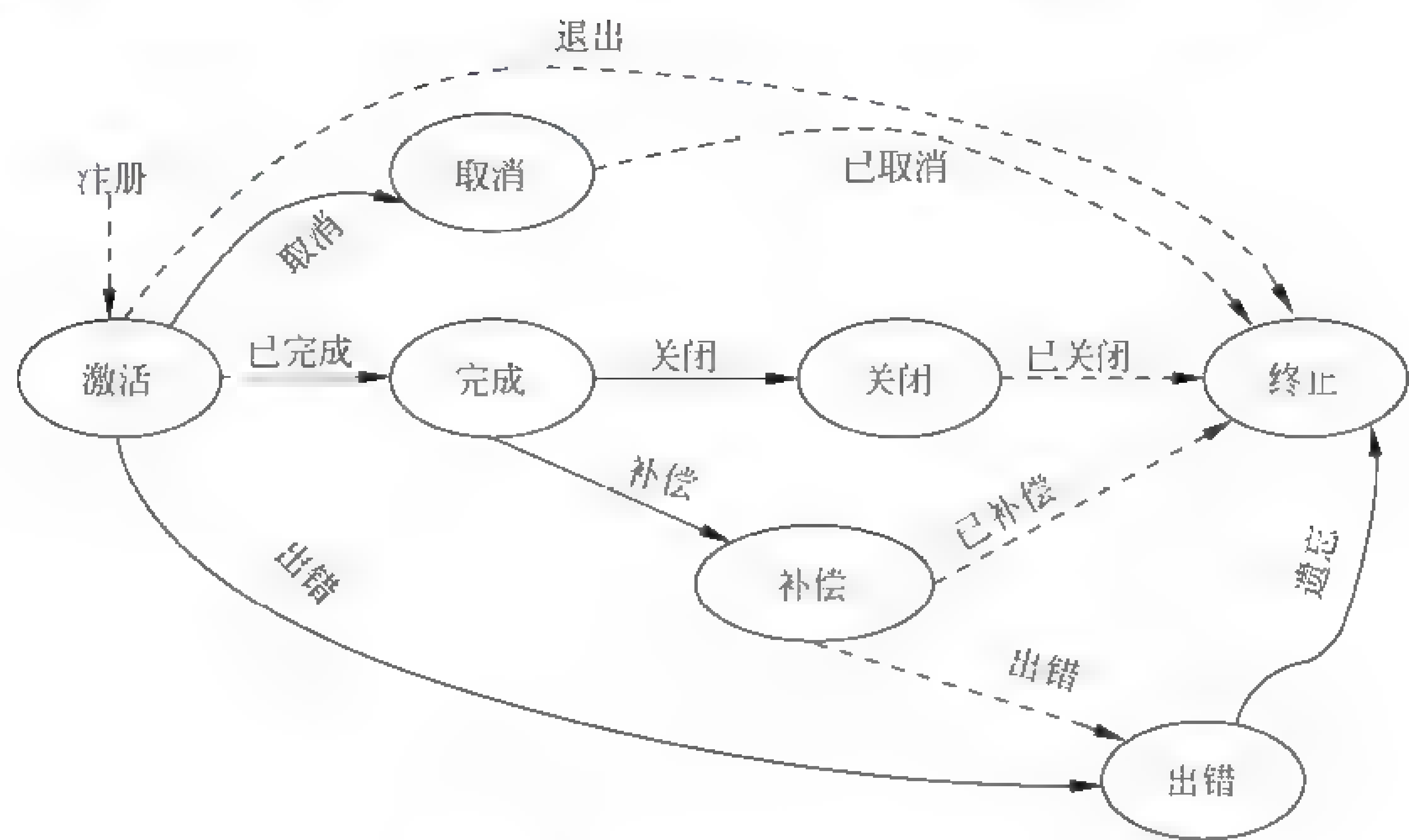

图 3-55　业务流程执行状态图

表 3-17　ESB 与 BPEL 的区别

	ESB	WS-BPEL
功能	1．消息路由：将传入消息发送到目的地，该目的地通过硬编码方式连接的逻辑确定或基于内容的动态方式确定。路由是启用服务虚拟化的关键功能。在调用方和服务之间建立中间层可以在调用方不知道更改的情况下移动服务的位置 2．消息转换：将传入消息从一种格式转换为另一种格式。例如，可以将逗号分隔的消息转换为 SOAP，这样可以将数据传递到 Web 服务 3．协议中介：传入消息使用不同的协议从发出位置发送 4．事件处理：事件的传入消息一般通过发布和订阅模型分发给许多端点	1．业务流程：流程可以是有状态和长时间运行的，或者是事务型。长时间运行的流程无法像事务型那样回滚，不过，它们可以使用补偿处理程序撤销先前执行的活动。流程可用于实现组合服务 2．人工任务：业务流程的一个关键部分是能够将人员引入该流程。人工任务管理器启用一些步骤，通过这些步骤可以将人员作为一种服务来调用。工作流模式是使用 BPEL 扩展，然后通过外部（参与）任务或内联任务进行支持的 3．业务规则：集成的规则引擎允许创建和评估业务规则，而不是将决策硬编码连接到业务流程。授权用户可以使用 Web 浏览器更新该规则。同时，管理员可以激活更新，并将其导出，因此开发环境可以与运行时保持同步 4．WS-BPEL 能与 ESB 集成 5．应用 WS-BPEL 可以使 SCA 接口映射到可用于调用其接口与调用组件不同的服务
优点	1．处理消息：消息的传入和传出也许会用到协议或格式中介。当这些需求明显需要处理消息时，使用 ESB 可以提供许多优势，其中包括在转换中处理较复杂事务的能力。当这些需求需要使用 ESB 基本功能（如消息路由、转换或协议中介）之一时，则 ESB 是最佳选择 2．性能：ESB 在计划上能够处理大量的消息。例如，如果需求是每天处理 200 000 条消息，则 ESB 显然是较好的选择	1．BPEL 引擎的主要优点是能够编排业务流程。如果需求是处理具有复杂逻辑的流程，则 BPEL 是较好的选择。WS-BPEL 包含容器活动，如 ESB 不支持的 while 循环和范围。ESB 中的逻辑通常非常简单，而 WS-BPEL 可以处理更复杂的情形 2．WS-BPEL 引擎的另一个优点是能够让业务流程长时间运行并维持其状态。当需要状态时，不应使用 ESB，因为维护状态会占用许多自定义代码。如果需求是进行有状态的处理，则在选择时可以排除 ESB

续表

ESB	WS-BPEL
相似点	都可以与适配器一起工作，都可以转换数据，都支持组合服务模式
选择基本要求	1．如果需要有状态流程，则可以立即排除 ESB 2．如果要求在短时间内处理消息转换，则显然应该选择 ESB 3．当需求是以数据为中心的，则显然要选择 ESB 4．如果需求是以流程为中心的，则显然要选择 WS-BPEL

3.3.4.6 一个 ESB 应用的例子

一个实际 ESB 项目实施的成败，不仅要熟悉 ESB 的各种基础知识和相关方法，还要确定 ESB 能否按需求进行独立开发，即要熟悉 ESB 产品的配置、开发及优化操作，还需要制定正确的、量体裁衣式的解决方案，以及在熟悉 ESB 的情况下，独自研发 ESB，并且需要借助科学的项目实施方法论，从需求分析、方案设计、产品开发、测试、上线运行等各个方面进行全面的考虑。下面以航空公司 ESB 案例进行解析[①]。

通过 ESB、接口适配器、服务注册管理等整合技术，实现将企业内部现有的各应用系统之间的信息共享，提高企业内部应用系统的数据共享和交换效率，提升企业在市场上的综合竞争力和客户服务质量，是所有企业的一个典型需求。下面将以某航空公司的案例为基础，说明采用 ESB 整合航空公司电子商务、常旅客、航班动态、呼叫中心等系统的解决方案。国际和国内的主要航空公司内部也分布着众多已建和在建的用以支撑业务运行的 IT 系统，这些系统之间缺乏对信息共享性、系统兼容性和接口标准规范的统一考虑，造成系统之间的连接比较困难，应用和数据无法得到全面共享，且系统间“蜘蛛网状”的连接普遍存在。随着新系统的不断建设，在业务与流程方面的整合将会因系统和业务领域间的信息沟通障碍而面临越来越多的困难，以及对航空公司的整体发展战略带来制约。表 3-18 所示是航空公司电子商务与外围系统集成举例。

表 3-18 某航空公司电子商务与外围系统集成举例

需要集成的系统	待交换的数据	通信协议	数据格式
定座/离港（GDS/DCS）	Booking Fare Query Available Query	TTL: TCP/IP Others: TCP/IP	TTL: MATIPOthers: XML
运价管理系统	Fare Query	JMS	N/A
常旅客系统	SSO User information Award Redemption	HTTP	SOAP (Web Services)
电子支付系统	Authorization Payment Cancel	HTTP	XML

① http://www.ibm.com/developerworks/cn/websphere/library/techarticles/0905_loulj_esb1/

续表

需要集成的系统	待交换的数据	通信协议	数据格式
Fare Management	Fare Rule	N/A	N/A
呼叫中心系统	Booking Searching	HTTP/MQ	XML/SOAP
国际联盟系统	Dynamic Flight Information	HTTP	SOAP (Web Services)
客户主数据系统	Customer information	Web Service	XML/SOAP

在图 3-56 中，给出了某航空公司的一个 ESB 示例。在这个例子中，可以看到其电子商务系统、航班信息发布系统、客户主数据系统都是采用 Web Service/实时/XML 接口；呼叫中心采用 socket/实时/文本、WebService/实时/XML 接口；常旅客系统采用 FTP/批量/文本、WebService/实时/XML 的接口；大客户系统采用 Database 的接口形式。

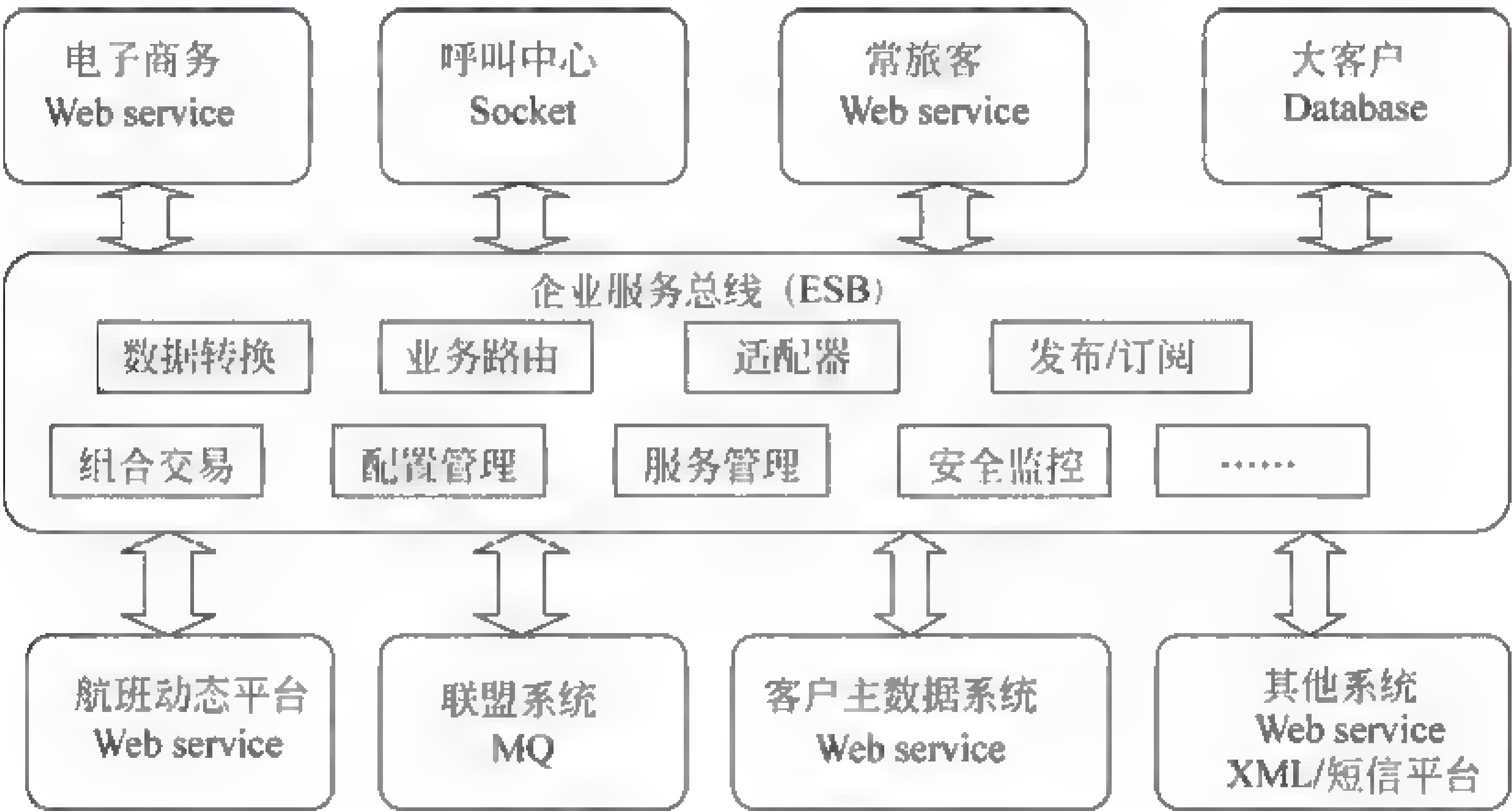

图 3-56　航空公司 ESB

由于基于接口的数据格式通常是不同的，与 ESB 相关的系统可以分为以下两类：

1．基于 XML 报文的应用系统

基于 XML 报文交互是比较理想的方式，是目前业界较为推荐的标准方式。需要说明的是，尽管都采用 XML 标准，由于各个系统的需求差别和已经建设周期的不同，使得不同的应用系统采用的 XML 消息很难完全兼容，这时就需要由 ESB 实现相应的转换。

2．基于专有报文/自定义报文的应用系统

基于专有报文的应用系统，如国内的订座系统，可以先保留现有的报文格式，由 ESB 实现现有格式与其他报文格式，以及采用 XML 格式在它们之间的转换。随着未来条件的成熟，这些系统逐步过渡到通过 XML 实现与 ESB 和其他应用系统的集成。

由于基于接口的通信协议的不同，与 ESB 相关的系统可以分为以下四类：

1．基于 Web Services 的系统

基于 Web Services 的系统，例如目前的呼叫中心和电子商务系统都可以提供这种方式，

可以使用 SOAP/HTTP(S)与 ESB 实现整合。

2. 基于 FTP/Socket 的应用系统

需要通过 FTP 交换数据的系统，如 FFP 系统等。ESB 可以直接支持 FTP 的方式。ESB 缺省提供文件适配器，其中就可以支持本地文件和远程文件通过 FTP 方式的读写。

3. 基于数据库的应用系统

基于数据库的系统，如大客户系统、数据仓库系统，可以通过 JDBC 适配器与 ESB 集成。

4. 基于传统应用连接的系统

对于这类系统可以通过定制的 Adapter 与 ESB 及其他应用实现整合，该 Adapter 可以以 Java 实现。另一方面，也可以通过 XML/MQ 实现与 ESB 的集成，这时，这些传统应用系统将调整为面向消息的方式。使用 MQ 作为一个通用的 Adapter 与 ESB 及其他应用实现整合，消息的格式可以逐步由现有的专有报文转变为基于 XML 标准的报文。

在表 3-18 中，粗略列举了某航空公司的电子商务系统与其各主要相关系统间交换的业务数据内容，以及通信协议和数据格式。从表中可以看出其复杂性，如果没有一个统一的集成平台的支撑，那么数据格式转换、通信适配器的开发、传输可靠性保证等问题都需要依赖于自主开发，其风险是不言而喻的。表中的数据模型如图 3-57 所示。

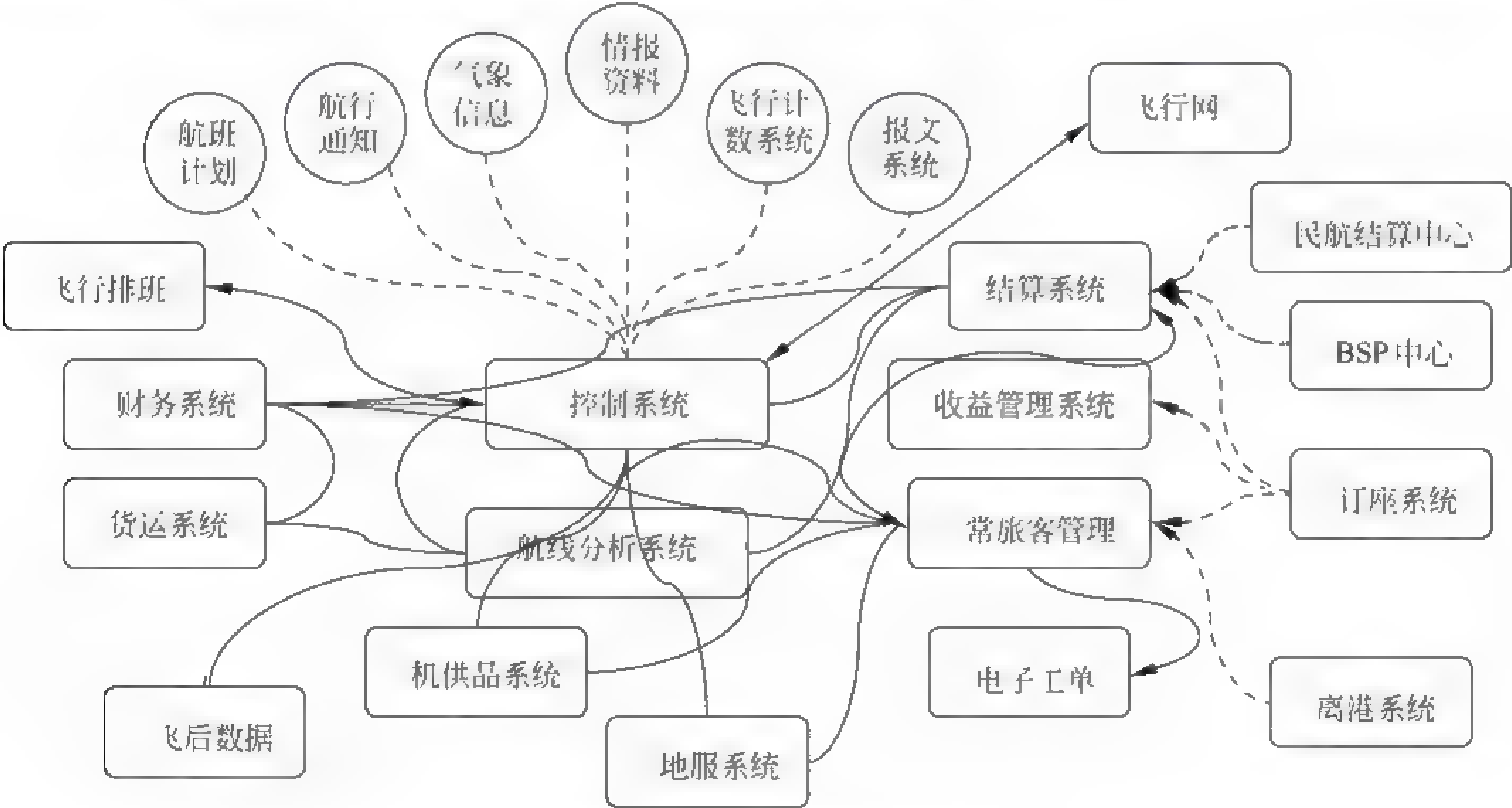

图 3-57　航空公司的数据模型

3.3.5　面向服务体系架构建模语言（SoaML）

SoaML（Service oriented architecture Modeling Language）是一个扩展 UML 2 的 OMG 标准，用于建模服务，面向服务架构（SOA）和面向服务的解决方案，如图 3-58 所示。此模型已在 IBM Rational Software Architect（SOMA）中实现，如图 3-59 所示。表 3-19 所示是 SOMA 服务识别技术。

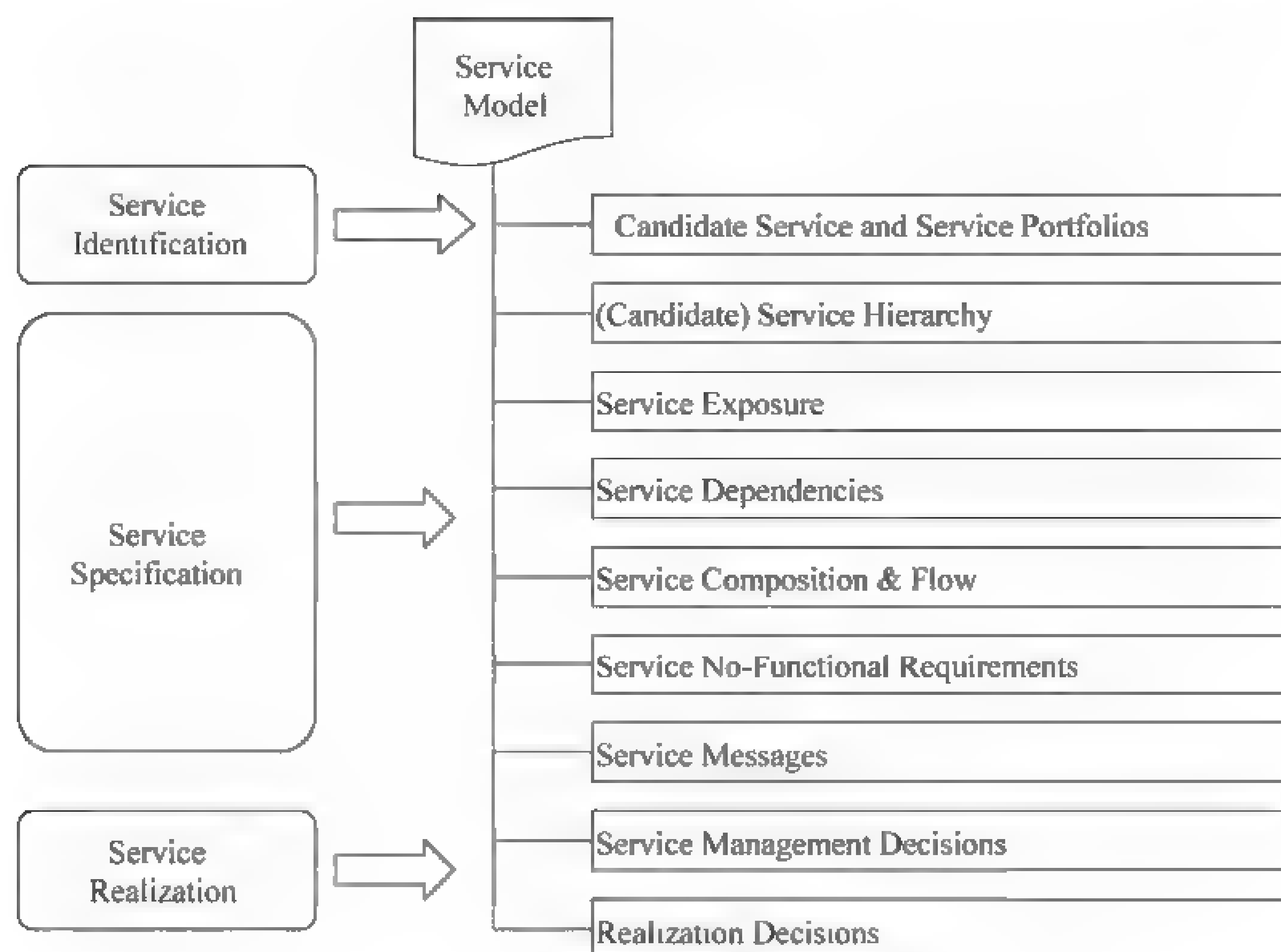

图 3-58　SOMA 服务模型各个方面

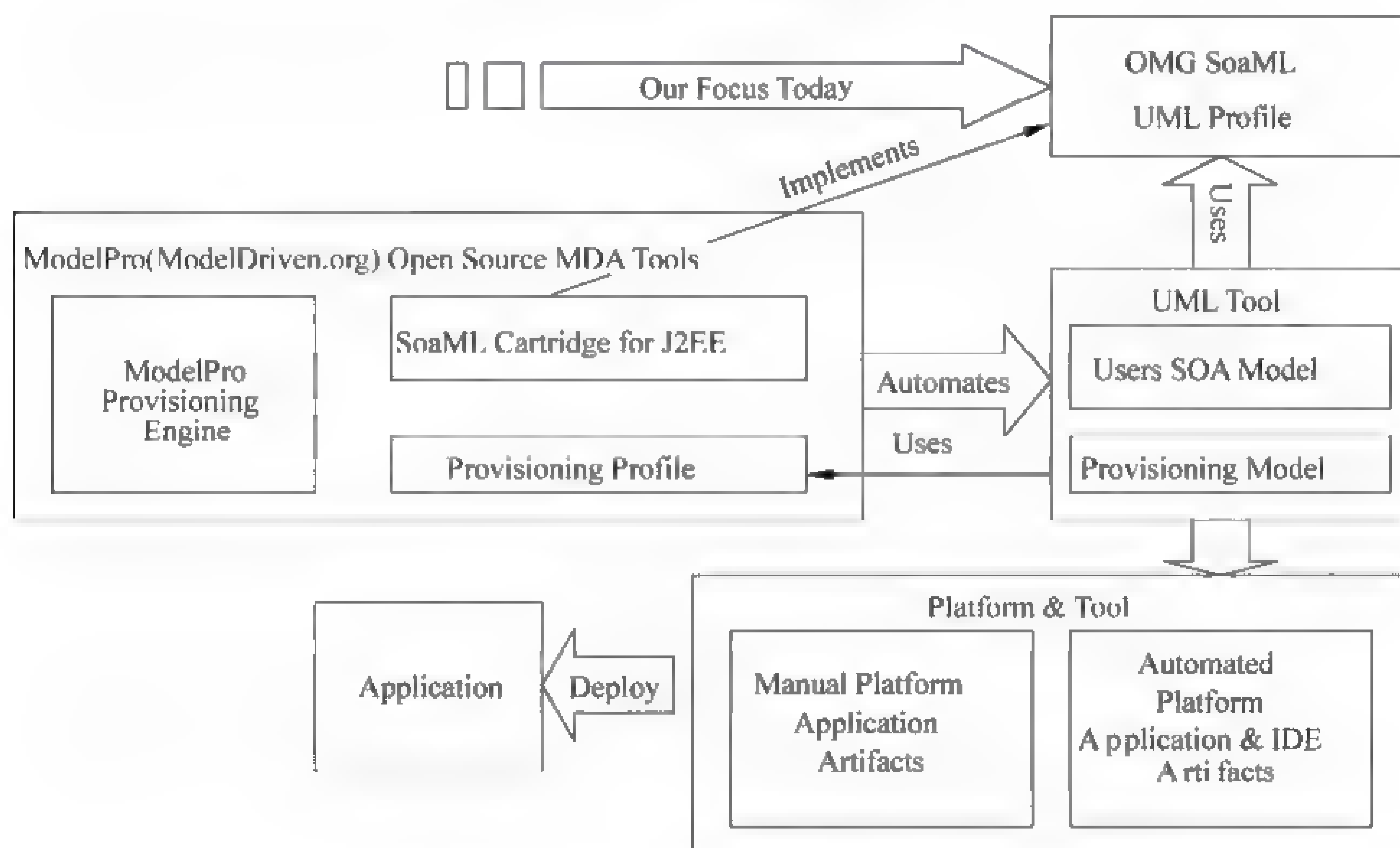

图 3-59　以 SomML 为支持的模型驱动 SOA 建模结构

表 3-19　SOMA 中的服务识别技术

技术	目标
目标服务建模	保证每一个考虑的业务目标都由一个候选服务支持，而且每一个识别的候选服务都至少支持一个业务目标
功能模型分析	使用组织功能结构的静态模型来识别候选服务，每个业务单元都需要或提供这个业务功能
业务过程分析	通过将候选服务映射到业务过程模型中所发现的业务任务、角色和过程来识别这些候选服务

续表

技术	目标
业务规则分析	识别封装了可用业务规则的候选服务
业务用例分析	从业务用例实现中识别候选服务
域模型分析	识别管理关键信息元素所需要的候选服务
现有资产分析	识别可以从现有 IT 软件资产获得的候选服务

3.3.5.1　SomML 基础

SoaML 是针对于服务的 UML Profile 和元模型的规范。SoaML（建模语言）是对 UML 2 的一个标准扩展，其目的是为了简化服务的建模。以下概要说明了该标准的必要性及目的。

服务是通过一个定义良好的接口而提供的功能，并且社区可以获得它。SOA 是一个架构范型，它定义了人、组织和系统如何提供和使用服务去获得预期结果。而 SoaML 为使用 UML 构架和建模 SOA 解决方案提供了一种标准手段。Profile 使用 UML 内建的扩展机制，同样，SoaML 也 Profile 机制（如图 3-60 所示），它是根据现有 UML 概念来定义 SOA 概念。SoaML 能和当前 UML 工具一起使用，但某些工具可能会提供增强、特定于 SOA 的功能，并对兼容的 SoaML 元模型提供支持，如图 3-61 所示。

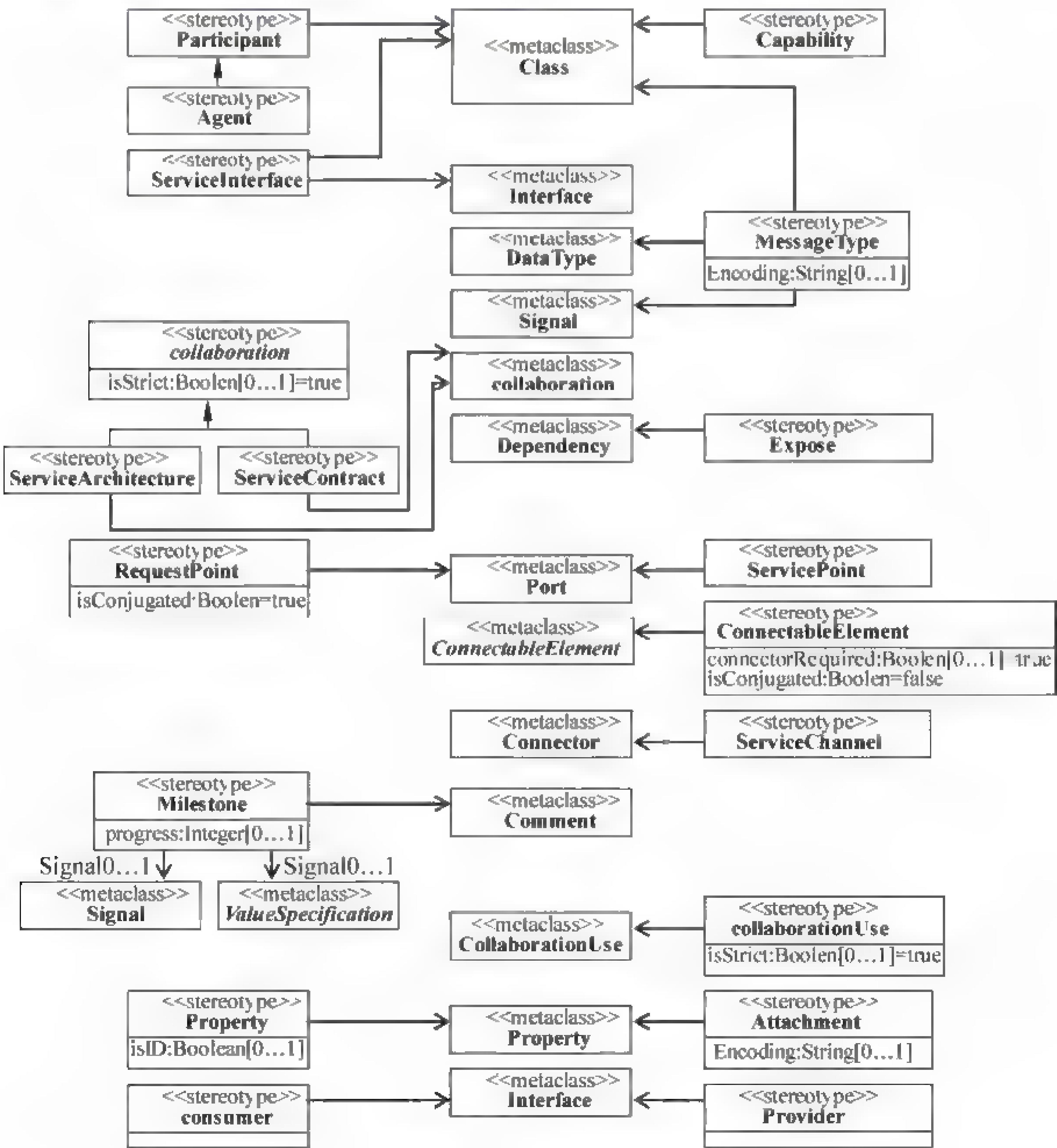

图 3-60　SoaML Profile

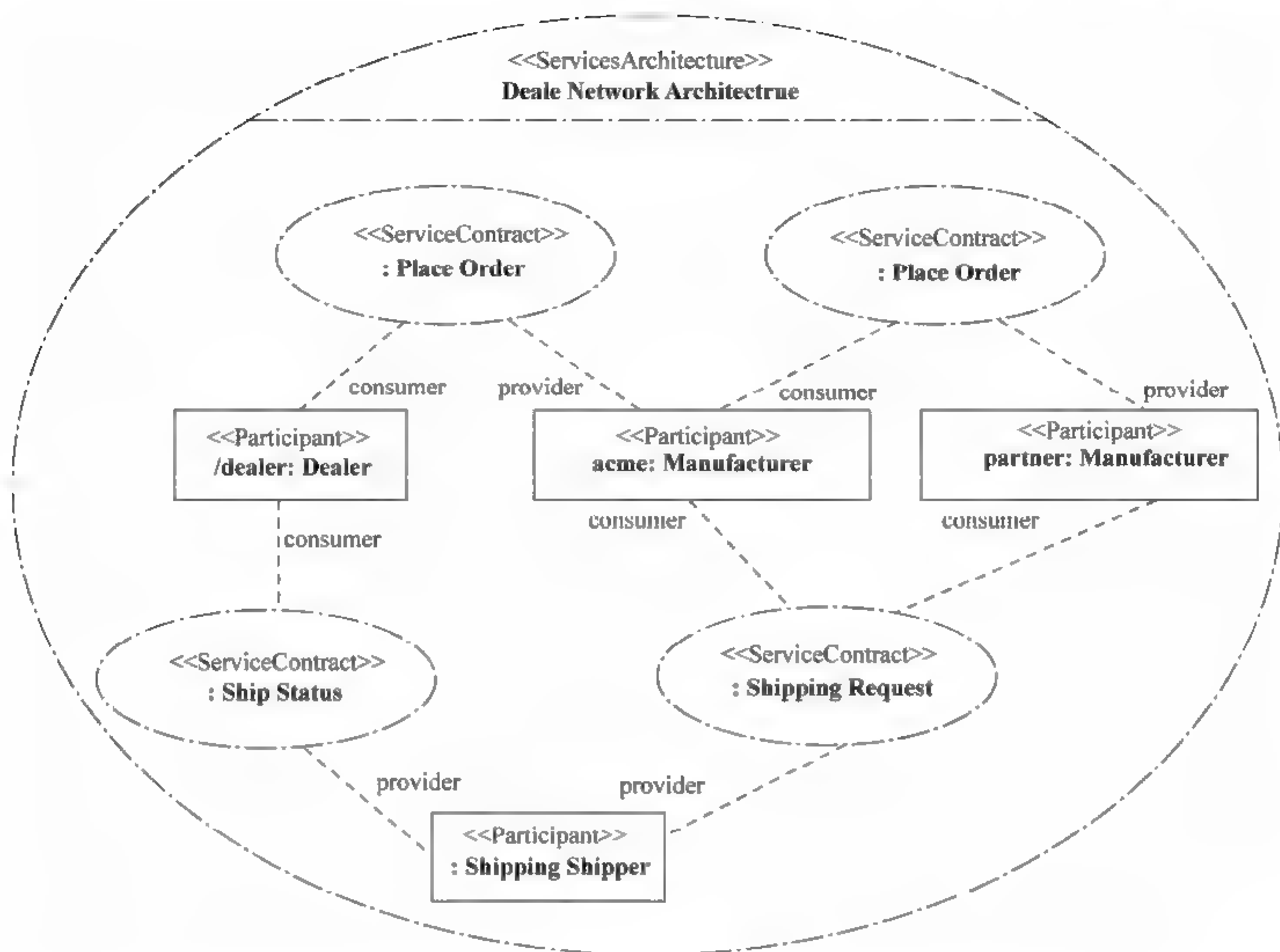

图 3-61　SoaML 联系图

建模和模型驱动的开发（MDD）可以帮助实现业务需要间联系。且模型允许从执行的具体过程中抽象出来，并关注于驱动业务和结构性决定的问题。在一定的程度，将要描述的方法，对业务和方案开发应用了一种基本性的原则：关注点的隔离和耦合的解离。为了这个目的，SoaML 就诞生了。SoaML 是一种对象管理组（OMG），它用于消除这个隔阂，并帮助实现 SOA 的潜力。SoaML 是对 UML 的小型扩展，以支持 SOA 建模。它提供了 SOA 的抽象，以关注描述参与者的需要和功能，并将它们联系在服务价值链中。

下面是 SoaML 的基本内容要点：

（1）标识服务，它们要实现的需求和它们之间的依赖关系。

（2）描述服务，包括它们提供的功能和在消费者、提供者之间交换的协议和数据。

（3）定义服务消费者和提供者，以及服务功能是如何与服务规范协议和要实现需求的一致性，并且定义了消费者所使用服务与提供者服务间的实现方式。

（4）使用和提供服务的策略。

（5）能够定义分类模式（它包含了对大范围架构提供支持的内容），组织性和物理分区模式及约束。

（6）定义服务和服务使用需求，并将它们关联到相关的 OMG 元模型，如它们实现、支持或完成的 BMM（Business Motivation Model）行动方针、BPDM（Business Process Definition Metamodel）流程、UPDM（Unified Profile for DoDAF and MODAF）运营能力和（或）UML

用例模型元素。表 3-20 所示是 SoaML 与 BPDM 的映射关系。

表 3-20　SoaML 与 BPDM 的映射关系

SoaML 概念	BPDM 概念
Interface & Service Interface	Interaction Protocol
Message	Type
Participant	Processor Role
Role in service contract	Interaction Role
Service Contract	Interaction Protocol
Service Interface	Implied by interaction protocol
Service Port	Involved interaction association
Service Realization	Processor Role
Service Contract Use	Interaction
Services Architecture	Proces

（7）当前 SoaML 关注的是基本服务建模概念，其目的是把这些概念作为进一步扩展的基础，这二者都是与集成其他 OMG 元模型（如 BPDM 和即将到来的 BPMN 2.0，以及 SBVR、OSM、ODM 等）相关的，如图 3-62 所示。

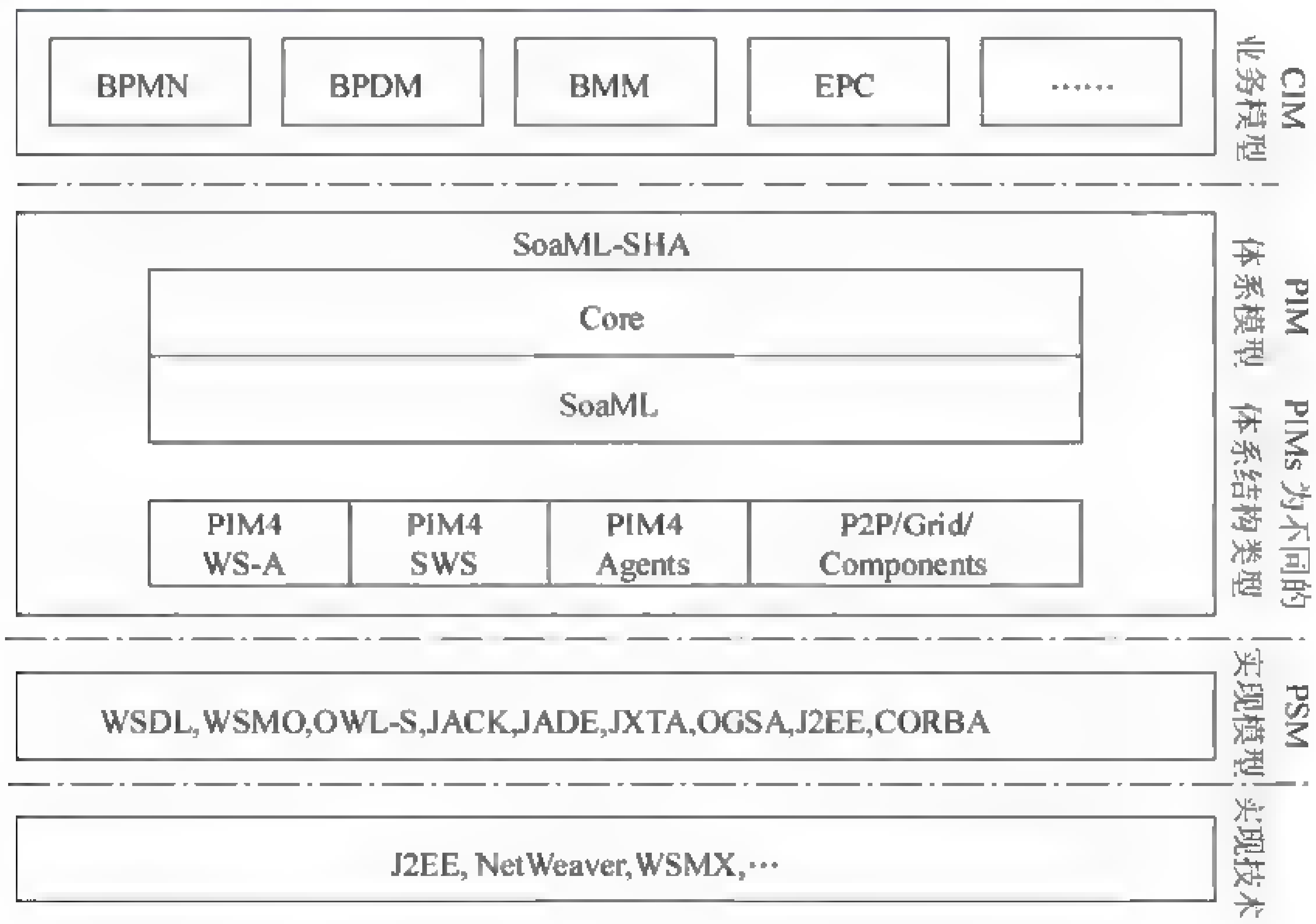

图 3-62　SOAML 结构

SoaML 是针对服务的 UML Profile 和元模型的回应，并且 SoaML 是在 OASIS SOA 参考模型框架内发挥作用。SoaML 依赖模型驱动架构（Model Driven Architecture）来将业务、系统架构和企业设计映射到支持 SOA 的实现技术（如 Web 服务或者 CORBA）。但其关注点主要是业务和架构。从而使得 SoaML 面向业务和面向系统的服务架构能相互合作来支持企业任务，如图 3-63 所示。

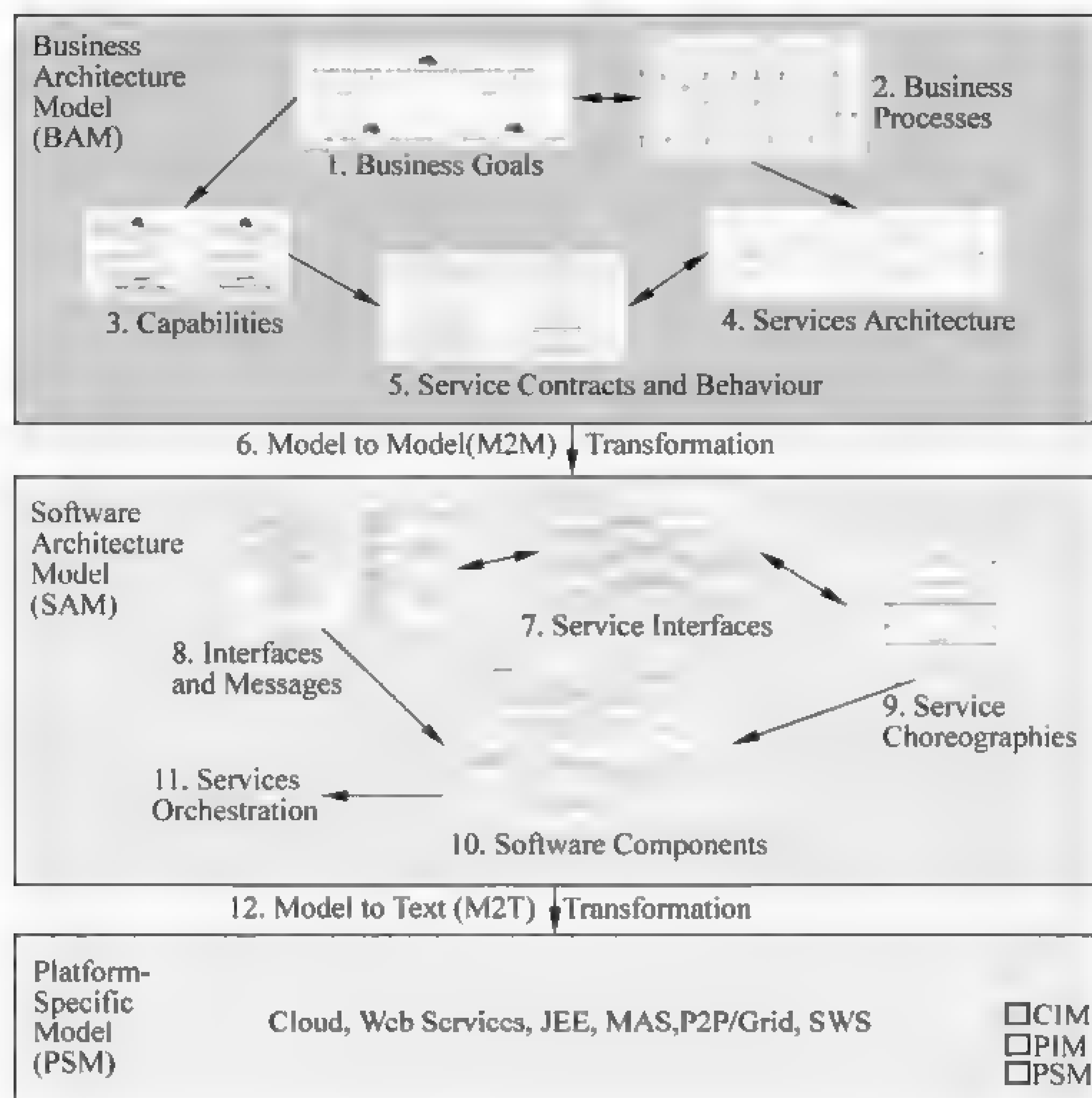

图 3-63 面向 SoaML 的 MDA 方法

除此之外，SoaML 还具有以下一些优势：

（1）支持模型层次上的交互性与集成性。

（2）提供了隔离于平台多样性，以及低层次 Web 服务的 XML 文件标准之外的高层次抽象性。

（3）通过使用结构作为业务需求与自动化 IT 方案之间的桥梁，来处理业务集成与服务交流的关注点。

（4）通过模型驱动的结构（MDA）来支持已存在平台之上或者之间的 SOA。

（5）支持灵活的平台选择。

（6）解去方案结构平台执行的耦合，以阻止已存在的方案对平台发展造出不良影响。

（7）为末端到末端生命周期开发与管理平衡、集成已存在的 OMG 标准。

3.3.5.2 一个面向 SomML 的例子

SomML 例子采用一个包括独立的经销商、制造商和供货商组成的一个商业方案，并且将商业流程、规则和信息采用 SOA 的方式进行分析，即采用面向服务体系的建模。这个商业解决方案包括以下三个部分内容：

（1）基于 SOA 的经销商网络结构定义了一个 B2B 社区，如图 3-64 所示。图 3-65 所示是采购商服务，图 3-66 所示是订购商服务。

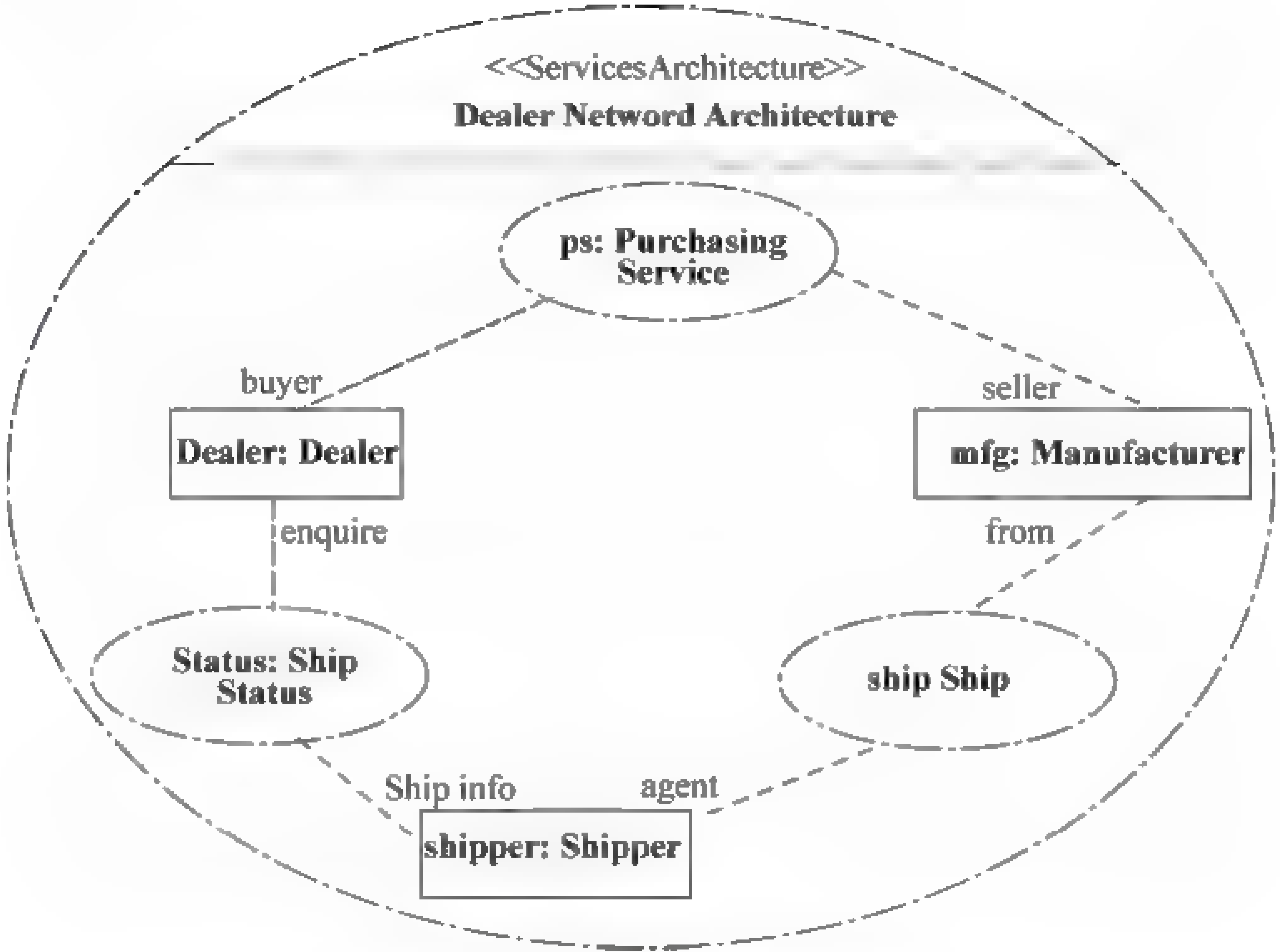

图 3-64　经销商网络体系结构

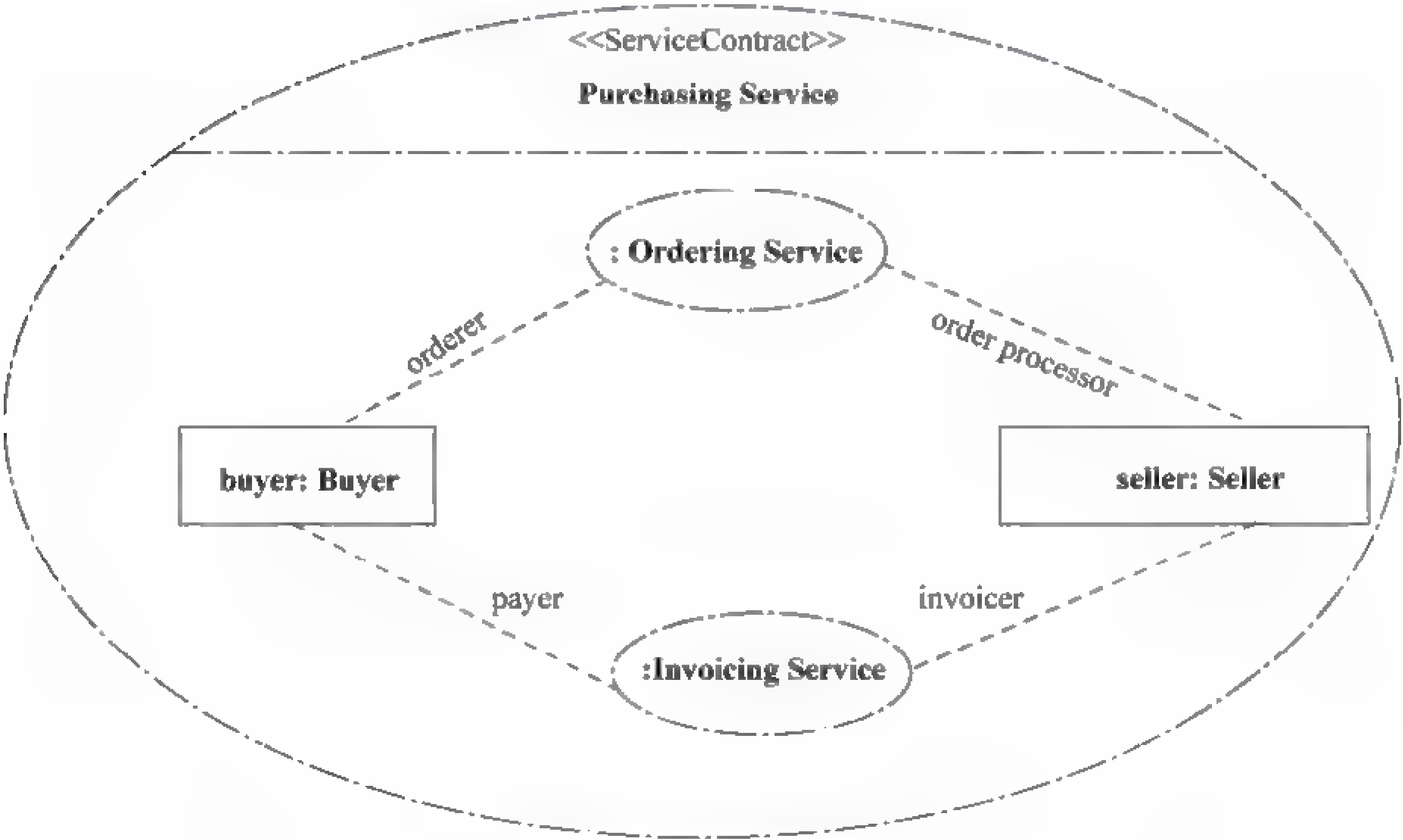

图 3-65　采购商服务

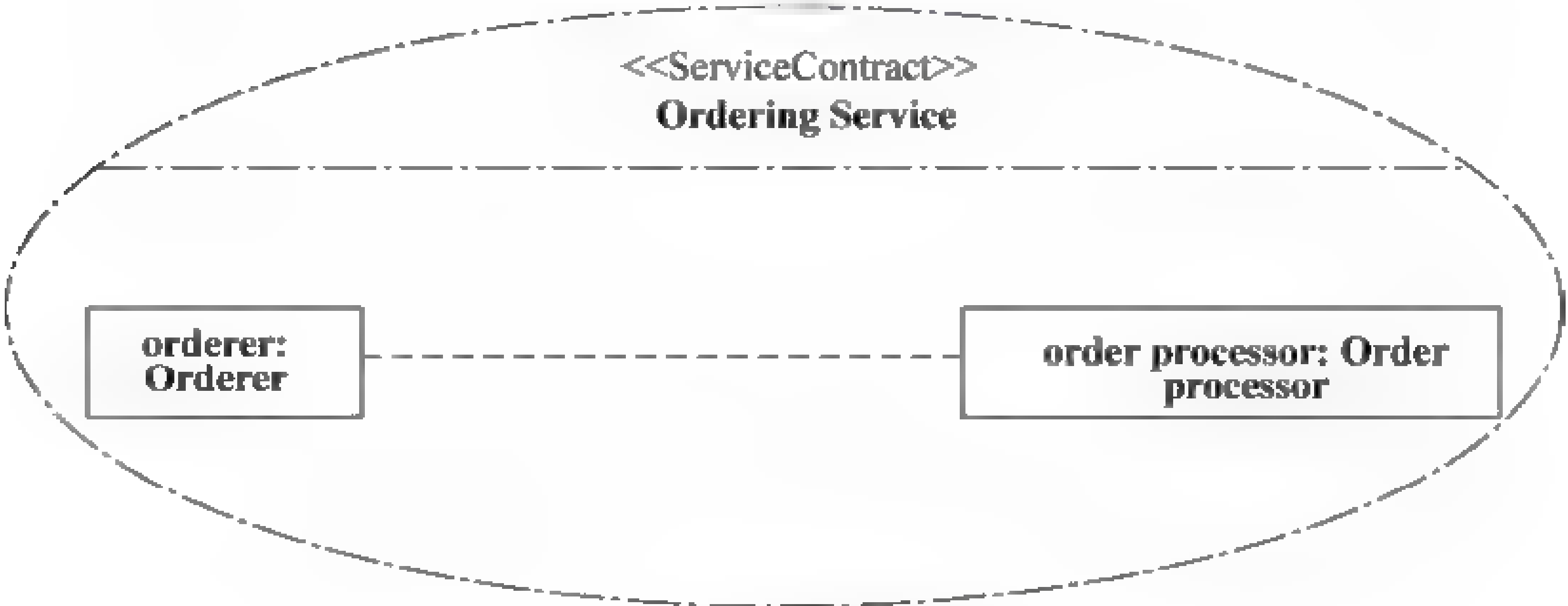

图 3-66　订购商服务

（2）为这个社区的成员开发一个支持内部服务流程的一个采购订单例子，图 3-67 所示是制造商组件，图 3-68 所示是服务接口，图 3-69 所示是作为服务合约的相关服务体系结构。

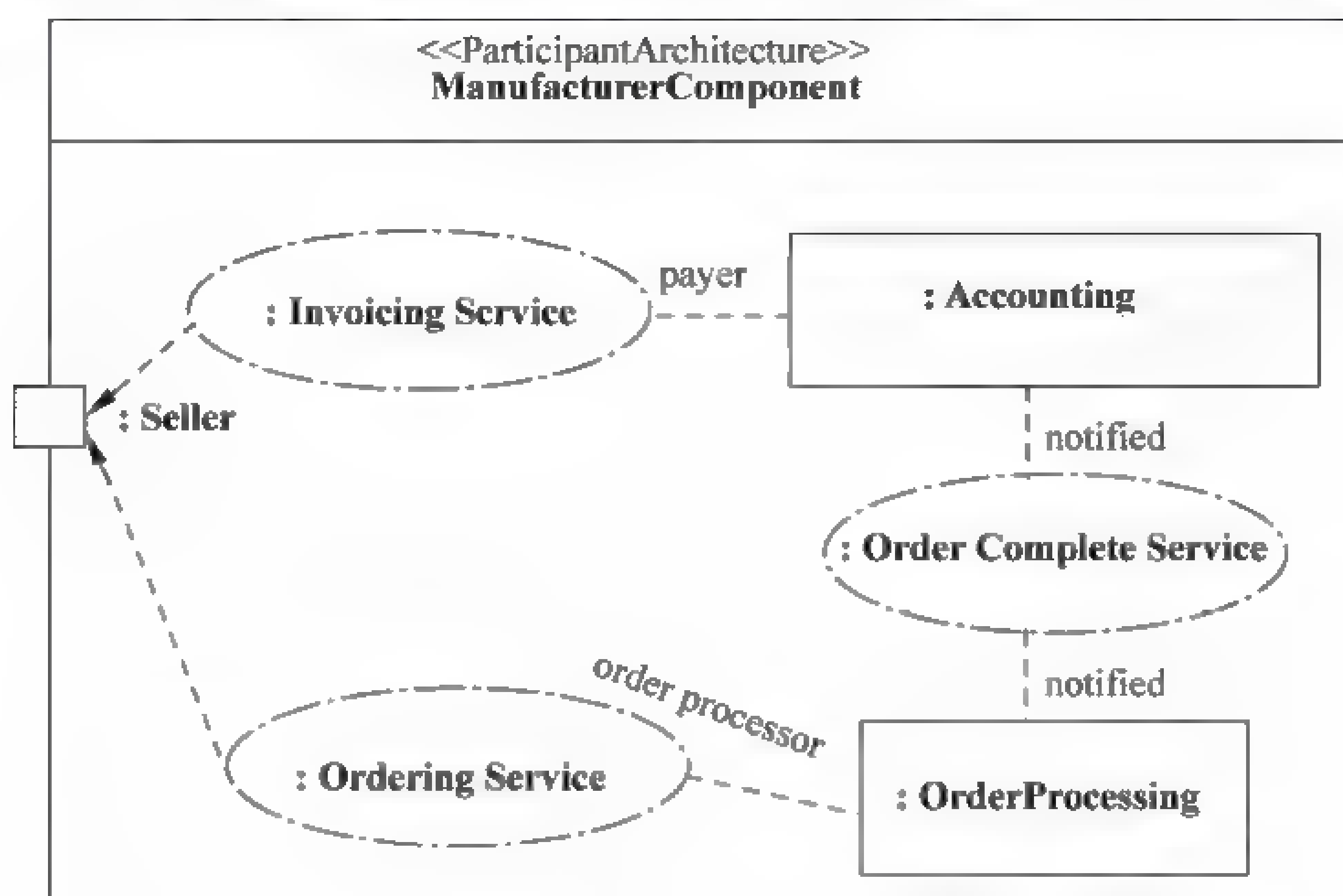

图 3-67 制造商组件

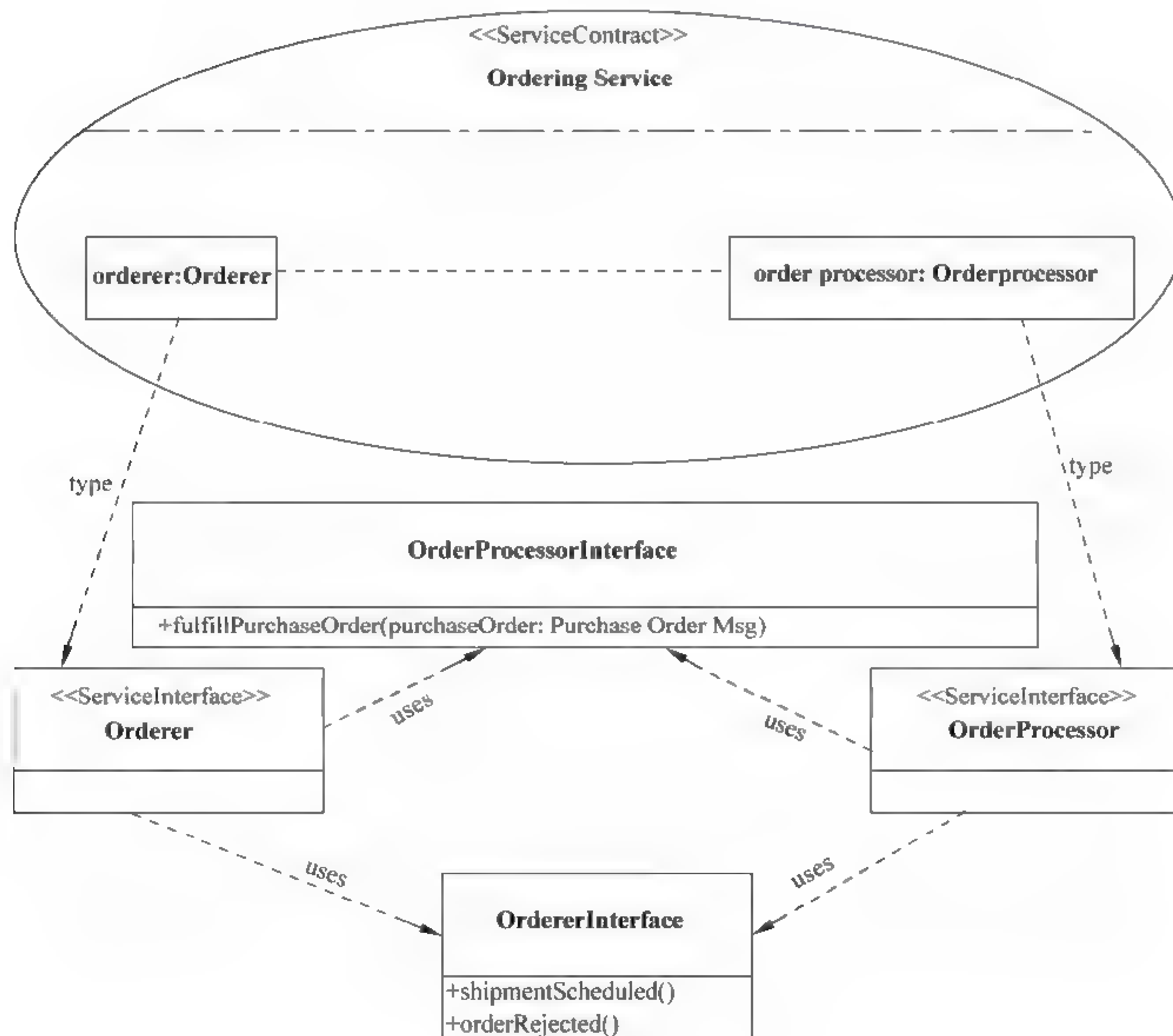

图 3-68 服务接口

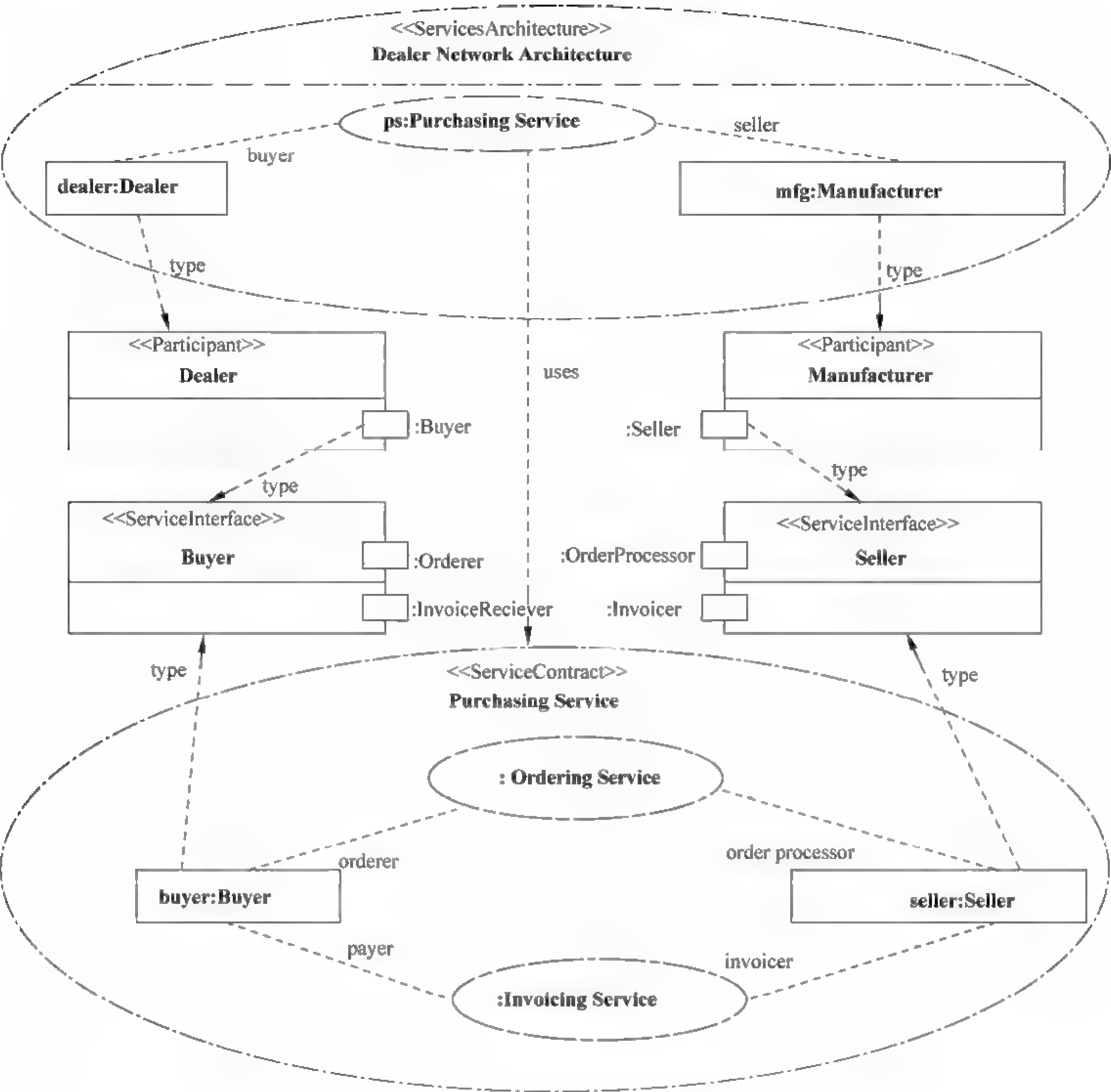

图 3-69　作为服务合约的相关服务体系结构

（3）样品采购订单数据模型需要适合社区的流程的例子。图 3-70 所示是流程的采购订单服务操作设计。

3.4　面向服务的软件语义化的软件分析设计方法

面向服务计算（SOC）和面向服务的方法一直都是学术界研究热点，具体体现在服务高效组合，但因服务自身存在处理语法、语义、识别、抽取等方面的问题，因此众多研究者提出了许多相关的方法进行解决，主要从形式化描述，智能计算，语义识别，逻辑推理等角度进行研究服务，直接表现在服务组合有效性、智能、自动化上，但大多只是从一定程度改进

了服务组合体系，未解决服务组合快速、精确组合。目前，语义服务组合是对传统服务组合更深层次的研究，采用基于 Ontology 的 OWL-S 和 WSMO 进行服务功能等多种语义的描述，主要聚焦在编制（orchestration）和编排（choreography），对传统服务组合模型处理语义等方面进行改进和增强，提高了服务组合效率，实现了不同程度的服务识别和抽取，并通过服务质量（QoS）衡量服务组合及服务的好坏，建立多种服务组合平台来实现服务组合。

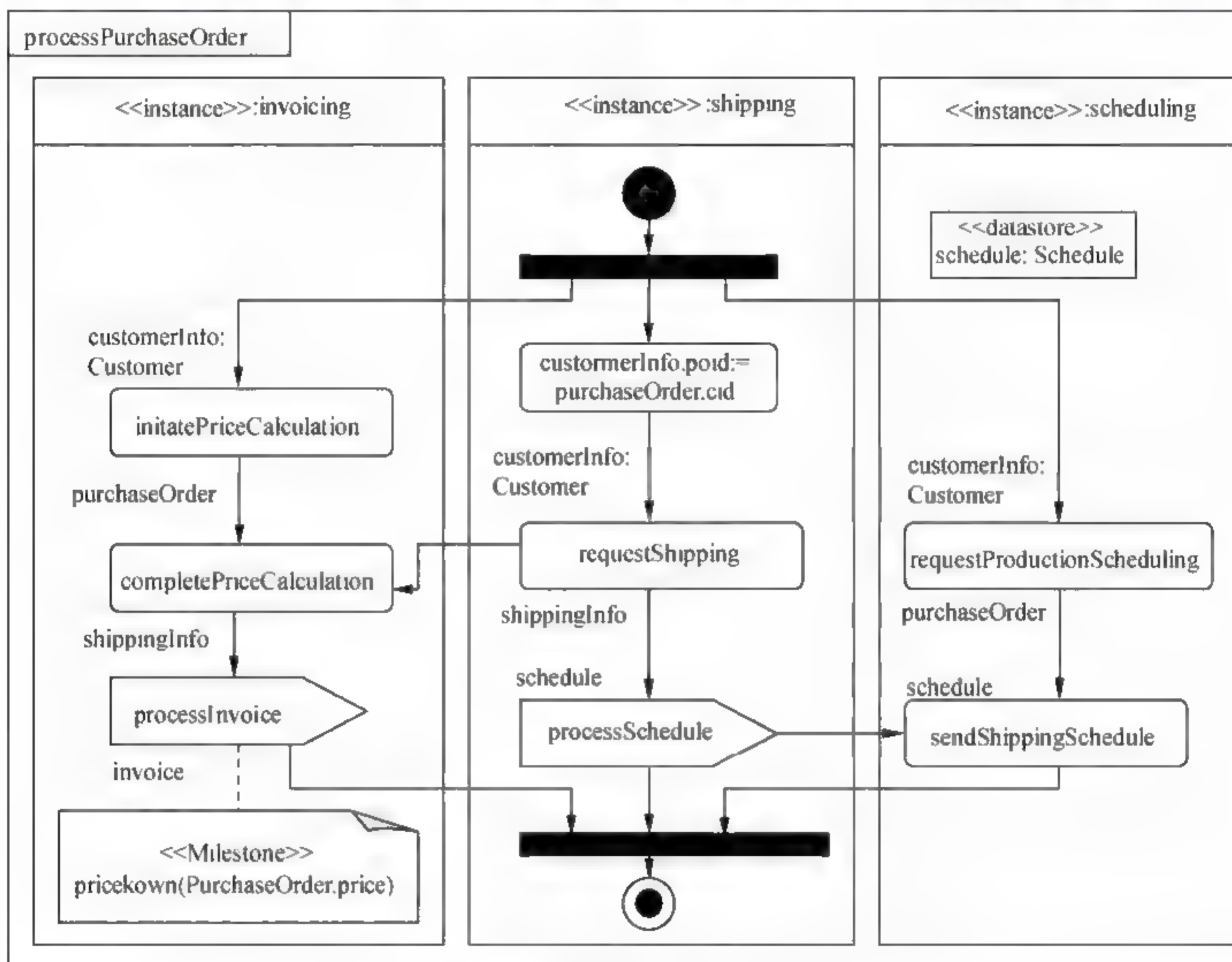

图 3-70　流程的采购订单服务操作设计

主题图是一种信息资源定位的本体（语义），但与 Semantic Web 有重要的区别：Semantic Web 用于概念形式化描述和分类，而 Topic Maps 用来实现索引和辞典构建过程的形式化[4]，因此为 Semantic Web 领域带来了高度的精确性。并通过使用语义特征（characteristics）实现主题图的主体，达到信息资源导航定位，以及组织信息资源，实现高效的知识管理，因此被称为信息管理和知识管理的桥梁，是信息世界中的 GPS 定位仪，因此，在门户网站、网络地图和图书馆知识管理中得到了大量应用。同时，主题图具有如下特性：

（1）具有组织巨大的信息体；

（2）能捕捉有组织的服务；

（3）能表示复杂规则和流程；

（4）支持机器学习；

（5）能管理分布知识和信息；

（6）能聚集信息和知识；

（7）具有信息资源导航定位能力；

（8）具有语义识别抽取能力；

（9）提供最大的结构柔性；

（10）具有较强的主题刻面分类机制。

因此主题图给分布在不同领域的服务准确聚集提供了方法，而传统的服务组合模型是以服务请求、服务响应的 WSDL 服务功能描述，UDDI 注册请求机制以及 SOAP 传输机制实现聚集服务；这样，让分散的服务聚合在一起，带有极大的困难，主要表现在服务聚集时间长，模糊性强，语义识别抽取能力弱，不一致验证缺乏，效率不高等缺点。但随着信息量日益增长，用户使用群增多，这种弱点还会进一步增强。

3.4.1 面向服务的软件语义化概述

Web 服务组合的业务逻辑实现主要通过 WSDL 描述和 UDDI 注册中心进行交互，服务消费和服务提供者通过 WSDL 获得具体的服务功能描述，通过 UDDI 注册和获取可用的服务。而 WSDL 只具有纯的语法规则，缺少语义上的识别；并通过 SOAP 完成业务交换和实现（Web 服务技术详细在后面的章节有详细分析）。为了增强服务组合的选择、识别能力，需要向 WSDL 中注入语义，Patil A 等人创建了一种本体到 WSDL 映射方法和具体的注释框架，进一步提高服务组合的松散耦合度。而主题图作为一种知识组织技术，提出了一种基于主题的元数据组织和描述方式，提供了语义级的数据导航和组织方式，是知识管理和信息资源管理的桥梁；作为一门知识表示语言，主题图能够满足语义网的发展要求，解决信息的发现性问题，具有知识表示以及多种知识组织方法的优点。因此，我们结合 Patil A 等人和 Nikita Ogievetsky 提出的思想构建一种服务的导航定位方法[1][2]，进一步实现面向服务的软件语义化方法在软件研发中的作用，并结合现阶段语义 Web 服务组合研究的优点以及主题图的信息组织能力建立一个服务组合导航定位模型；从而建立语义 Web 服务与主题图融合模式。因此，我们采取研究路线是：根据以 Ontology 为桥梁实现语义 Web 服务与 Topic Map 映射转换，就直接体现在 OWL-S 与 WSDL、TMDM 描述间的转化和映射上，而一个共同点就是怎样通过 URI 标识、定位到合适的资源。这时首先明白 OWL-S、WSDL 和 TMDM 间的关系，并分析相关的概念，找出融合的方法。其次，分析 OWL-S、WSDL 和 TMDM 的内在机理。最后建立它们的融合结构。图 3-71 是基于 Ontology 的语义 Web 服务与主题图间融合直观图。图 3-72 是融合后的服务组合结构图，在图中的融合层是表达 WSDL、OWL-S 以及 TMDM 间的融合规则。

语义 Web 是由 Tim Berners-Lee 等人于 1999 年提出对 Web 扩展的方法，其目的是对网络资源进行分析、推理和识别等，并于 2000 年正式提出语义 Web 的体系结构。基本思路是希望通过将 Web 上的知识表示为机器可以理解和处理的形式，从而对信息的检索和处理能更加准确和高效，最终具备自动识别资源的能力，达到智能级的认识。在语义 Web 的体系结构中，最重要就是有底层的统一资源标志符（Uniform Resource Identifier，URI）和统一字符编码

① http://www.ontopia.net/topicmaps/materials/rdf.html

② http://www.mulberrytech.com/Extreme/Proceedings/html/2005/Cregan01/EMI2005Cregan01.html

(Unicode)，主要是有效实现资源定位接口向具体的业务功能链接；还有就是 RDF、RDF Schema 和 Ontology 层，这一层主要完成语义识别、实现自动知识抽取，而目前描述 Ontology 的语言有 OWL-S 和 WSMO 等，其中 OWL-S 是对 RDF、RDF Schema 进行一步扩展，不但提供基于 XML 的语法级的检测，还提供了语义级的推理。而语义描述语言中先是由 RDF 来承担，然后到 RDF(s)，再到 DAML+OIL，最后发展到 OWL 以及 OWL-S。其中 RDF 主要由 RDF Data Model、RDF Schema 和 RDF Sytax 三部分组成，通过这三个部分就可以实现资源的语义描述信息。

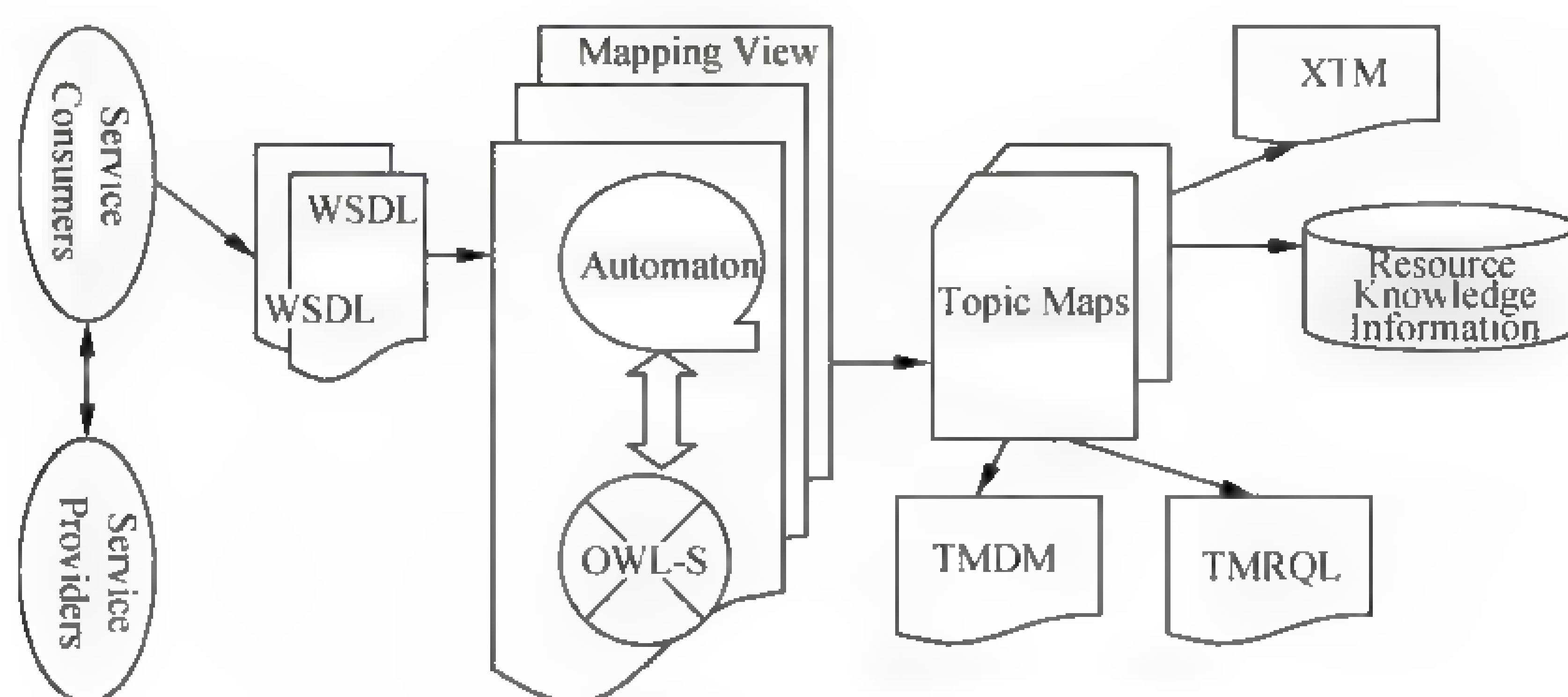

图 3-71　Web 服务与主题图融合直观图

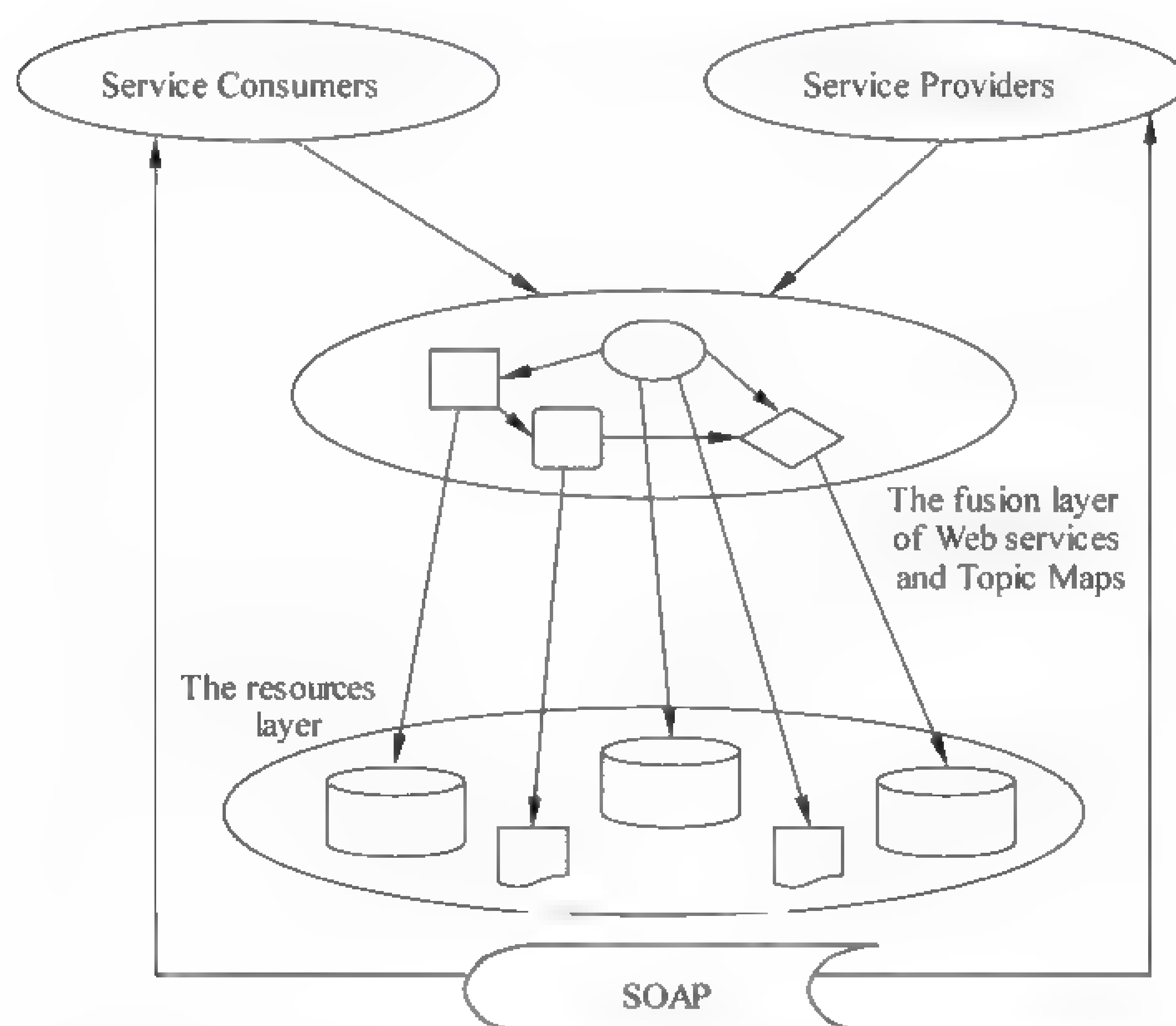

图 3-72　语义 Web 服务与主题图融合后的服务组合结构图

Web 服务也是 21 世纪初兴起的 WWW 新技术，因此就产生了语义 Web 与 Web 服务相结合的语义 Web 服务，而语义 Web 服务的目标是在 Web 服务的描述中加入足够的语义信息，使 Web 服务成为计算机可以理解的实体，从而支持 Web 服务的自动发现、自动组合和自动执行等，最终达到多样化的分布式计算。直接表现就是将语义 Web 的特征置入到 Web 服务

中去，实现 Web 服务各项工作的自动化，如图 3-73 所示。目前，基于 Ontology 的 OWL-S 和 WSMO 都对 Web 服务语义描述提供了方法。在 OWL-S 中，从 ServiceProfile、ProcessModel 和 ServiceGrounding 三个方面对 Web 服务进行了刻画。其中，ServiceProfile 类似于服务的黄页，描述了服务的功能及相关属性；ProcessModel 对服务的过程模型进行刻画，描述了服务是如何工作的；ServiceGrounding 将过程模型与通信协议及消息格式等联系起来，描述了如何访问一个服务。在 WSMO 中，从 Ontologies、Web services、Goals 和 Mediators 四个角度进行刻画语义描述，其中 Mediators 是贯穿整个语义 Web 服务编排和排列的核心。因此它的原理是从 Web Compliance、Ontology-Based、Strict Decoupling、Ontological Role Separation、Description versus Implementation、Execution Semantics 和 Service versus Web service 方面进行设计的，它们对比如表 3-21 所示。在服务消费与服务提供间查询 UDDI 服务信息，以及通过 WSDL 抽取具体的服务功能时，一般都是通过关键词方法查询和匹配较合适了的服务。因此国内著名学者史忠植等人引入了动态描述逻辑方法，来提高服务发现的查准率和查全率。

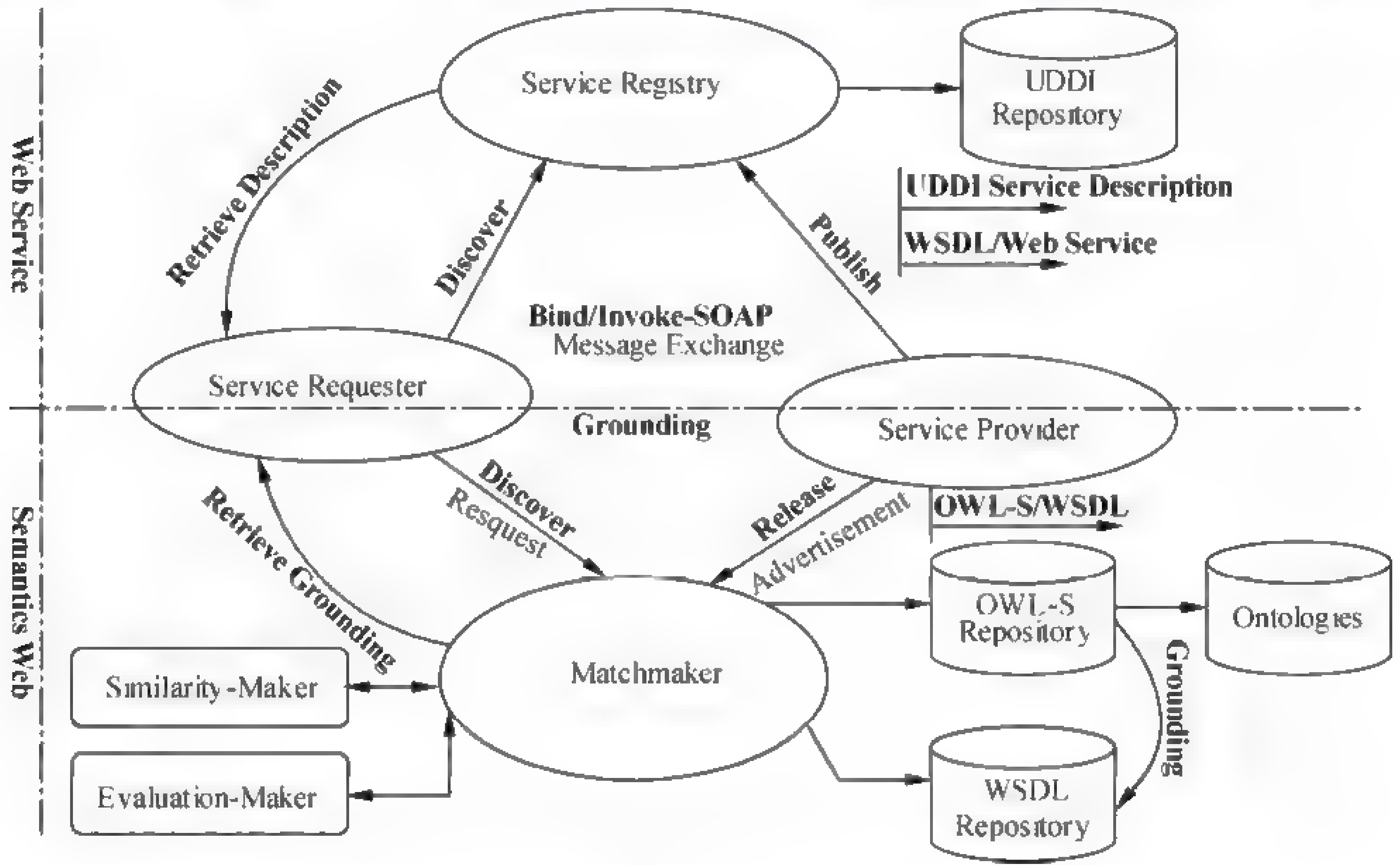

图 3-73 语义 Web 服务结构图

表 3-21 OWL-S 与 WSMO 的异同对比表

	OWL-S	WSMO	当前 Web 服务技术
服务发现	Profile	目标和 Web 服务（capability）	UDDI API
服务消费与聚集	过程（流程）模型	服务接口（编排+排列）	BPL4WS/WS-CDL
调用方法	Grounding+WSDL /SOAP	Grounding（WSDL/SOAP，基于本体）	WSDL/SOAP
异构处理		中介器	

而面向服务的语义化软件中的语义 Web 主题服务是在语义 Web 服务中置入主题图，提高 Web 服务发现的效率，并把这种置入后的名称称为语义 Web 主题服务（Semantics Web Topic

Services：SWTS），其主要把 OWL 作为 WSDL 与 TMDM 间的描述桥梁。因此，首先分析研究 OWL-S 与 WSDL、TMDM 的特征关系，然后用描述逻辑进行描述推理。如图 3-74 所示。最终不但要提高查询、发现和匹配的查准率和查全率，还要提高整个服务计算的效率，以及面向服务的语义化软件研发效率。

图 3-74　语义 Web 主题服务形成图

3.4.2　OWL-S 与 WSDL、TMDM 的特征关系

OWL-S 是用 OWL 语言描述的 Web 服务本体，而 WSDL 是对服务功能进行描述，WSDL 只有语法级的判断，虽然 WSDL-S 具有一定的语义，但 WSDL-S 不能有效表达数据和服务语义，这是因为 WSDL 是 OOP（Object Oriented Programming）的直接产物，因此，WSDL 文档只能反映 OOP 的内容，更不能去描述 Web 服务本体，却在具体的服务消费与服务提供者间的服务抽取是通过标准的 WSDL 去获得，但是面向服务的应用基本上还是集中在 WSDL 上。因此，就需要建立一种 WSDL 与 OWL-S 间的转化或映射的方法，这就是语义 Web 主题服务构造的第一步。而 XTM 主题图与 RDF 间互操作是存在问题的，因此，ISO/IEC 在 2008 年正式发布了 TMDM[①]，这为 OWL-S 与 TMDM 间的转化或映射提供了表示方法，并可以在 OWL-DL（Description Logic）中描述 Topic Map；这里，OWL-S 与 OWL-DL 的相同点是基于 OWL 的描述方法，不同的是 OWL-DL 是 OWL-S 的子集，OWL-S 表示用 OWL 去描述语义 Web 服务，而 OWL-DL 是 OWL 的一个子语言。这就是语义 Web 主题服务构造的第二步。既然 WSDL 可以通过 OWL-S 进行描述，主题图可以在 OWL-DL 中描述，则通过 OWL 就可以实现 WSDL 向主题图转化或映射，这就是语义 Web 主题服务构造的第三步。

OWL 定义了 OWL-Lite、OWL-DL 和 OWL-Full 三个不同的子语言，其中，OWL-Full 是 OWL 完整版语言，并全部使用 OWL 语言基元，并允许相关基元随意地与 RDF 与 RDFS 相互结合，因此，OWL Full 有最强的表达能力，但不对推理做任何保证；其优点是在语法和语义上与 RDF 完全向上兼容。OWL-DL 是 OWL-Full 的子语语言，且 OWL-DL 包括 OWL 语言的全部约束，但却也可置于特定的约束条件下，同时，在保证推理的完备性和可判定性的前提下，有尽可能强的表达能力，因此，其优点是允许有效的推理支持，缺点是与 RDF 弱兼容。OWL-Lite 进一步把 OWL-DL 约束限制到语言构造函数子集，因此，表达能力最有限，推理效率高，其优点是既容易掌握又容易实现。实际上，OWL-Lite 和 OWL-DL 特别相似且都是可判定的，而 OWL-Full 是不可判定的，但 OWL-Full 可提供最大的表现力和 RDF 没有计算保证的自由语法（如支持把类当做个体）。且每个合法的 OWL-Lite 都是合法的 OWL-DL，每个合法的 OWL-DL 都是合法的 OWL-Full，每个有效的 OWL-Lite 结论都是有

① http://www.isotopicmaps.org/sam/sam-model/

效的 OWL-DL，每个有效的 OWL-DL 都是有效的 OWL-Full 结论。同时，OWL-Full 可以看成是 RDF 的扩展，OWL-Lite 和 OWL-DL 可以看成是 RDF 的约束化扩展，所有的 OWL 文档都是 RDF 文档，所有 RDF 文档都是 OWL-Full 文档。因此选择在 OWL-DL 中描述主题图是取决于主题图约束可表示性和需求需要 RDF 的元模型机制。

描述逻辑（Description Logic，DL）是基于概念（类）和角色（属性）理念的知识表示形式，是构建在框架知识表示方法和语义 Web 的基础上。DL 是对概念化知识进行表示和推理的逻辑形式，可以看做是框架、语义 Web 和一阶逻辑的有机结合。使用特定的 DL 提供的概念和角色构造函数来建立原子概念（一元谓词）和原子角色（二元谓词）的表示式，用语义 Web 和框架等来定义 DL。同时，DL 是阶谓词逻辑的可判定子集，而 DL 系统能够提供可判定的推理服务，在 DL 中引入一阶逻辑不仅解决了语义问题，而且得到了推理机制；这样通过提供清晰的模型论语义，DL 就能够处理结构化概念的表示和推理。因此，语义 Web 本体语言 OIL，DAML+OIL 和 OWL 都建立在描述逻辑基础上，并且 Horrocks I 人证明了 OIL 与描述逻辑 SHIQ 等价、DAML+OIL 与描述逻辑 SHOIQ(D)等价以及 OWL Lite 与描述逻辑 SHIF(D)等价、OWL DL 与描述逻辑 SHOIN(D)等价[8]。因此，面向服务的语义化软件分析方法采用描述逻辑 SHOIN(D)来描述 OWL 与 WSDL 和 TDMD 间的特征关系。因此，WSDL、OWL、主题图融合关系如图 3-75 所示。

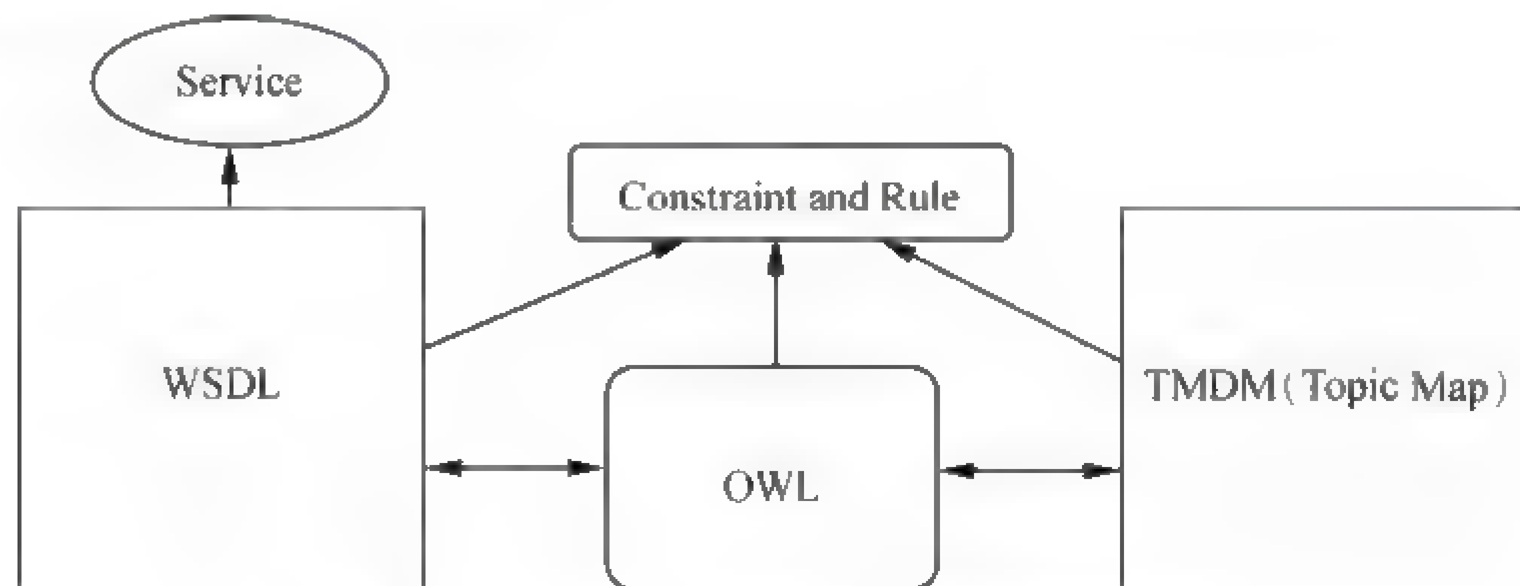

图 3-75　WSDL、OWL、主题图融合流程

3.4.2.1　OWL 与 WSDL

用 OWL 来填充 WSDL 的语义，除了 W3C 的定义的 OWL-S 标准提供的填充方法外①，还有针对 Ontology 向 WSDL 匹配映射的 METEOR-S 注释框架②，有基于该标准的 WSDL 向 OWL 转化的语义框架 WSDL2OWLS③，有一个面向应用的协助用户语义的 WSDL 注释程序集④，以及有还有一种基于 WSDL2OWLS 的 OWL-S 可视操作框架⑤，马里兰大学信息及网络动态实验室（MIND）开发的 OWL-S API 可用来具体实现语义 Web 服务的描述转换⑥，即将 WSDL 文档转换为 OWL- S 文档。它们共同的目标都是有效实现 WSDL 较强的语义性，通过

---

① http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/

② http://lsdis.cs.uga.edu/projects/meteor-s/

③ http://www.daml.ri.cmu.edu/wsdl2owls/；http://projects.semwebcentral.org/projects/wsdl2owl-s/

④ http://www.andreas-hess.info/projects/annotator/index.html

⑤ http://staff.um.edu.mt/cabe2/supervising/undergraduate/owlseditFYP/OwlSEdit.html

⑥ http://www.mindswap.org/2004/owl-s/api/

Ontology 去填充 WSDL 语义，采用 OWL 去向 WSDL 填充具体的语义。不同的是：它们采用了不同的方法去实现，且利用不同的语义表示去达到目的。

在 W3C 标准定义了 WSDL 与 OWL-S 间的映射关系，通过 Atomic Process 和 Operation 运行与 Inputs/Outputs 和 Message 动作关系作为一个中间并行过程，并将绑定到 SOAP 和 HTTP 等上的 WSDL 服务功能实施填充语义，这是采用 Process Model 和 DL-based Types 的过程的并行方法来实现的。

METEOR-S 注释框架是一种 WSDL 向 OWL-S 匹配映射的专门框架（MWSAF），通过在 Web 服务中增加 Data Semantics (semantics of inputs / outputs of Web services),Functional Semantics (what does a service do),Execution Semantics (correctness and verification of Execution),QoS Semantics (Performance/cost parameters associated with service)四种语义来增强 WSDL 的语义，但该框架主要聚焦在数据语义上。通过采用元素与结构两级模式实现 WSDL 与 OWL-S 间匹配，因此定义了 ElemMath 和 SchemaMatch 计算方法，其中 ElemMath 功能采用两个概念的名称 linguistic similarity 来实现，表明这两个概念必须要有两个相似的名称才能匹配识别。SchemaMatch 功能采用两个概念间的结构相似来实现，A concept in an ontology is usually defined by its properties, superclasses and subclasses，进而来发现语义。同时也能计算了概念相似和匹配，最得到 ElemMatch score and SchemaMatch score are then used to determine the final match score。然后分别采用平均概念匹配来获得 WSDL schema 和 Ontology 间的匹配概念相似度，并将这些计算结果用于决定映射作为框架的注释。而采用平均服务匹配来实现服务分类，且将这个计算作为 WSDL schema 和一个领域本体的所有概念的结果，结着将这些匹配结果作为 WSDL 的注释。

WSDL2OWLS 提供了 WSDL 和 OWL-S 间的转化方法，转化结果是一个 Process Model 和 Profile 的 Grounding 和 partial specification 的 complete specification，这是由于在 WSDL 和 OWL-S 中包含了不同的信息。WSDL 没有提供 Process composition information，因此使得 WSDL2OWL-S 转化结果缺乏 Process composition information；同时，WSDL 不提供服务性能描述，因此使得 OWL-S Profile 从 WSDL 生成也是必定的概要，并需要用手动去完成。然而，WSDL2OWL-S 的输出提供一种 OWL-S 描述的 Web 服务基本结构。所以可以得到该方法是基于 W3C 标准的，转化的效果不明显。

OWS-S API 提供了一组基于 Java 的 API 去完成 WSDL 向 OWL-S 转化，目前这些接口支持 OWL-S 1.0，OWL-S 0.9，DAML-S 0.7 等版本本体描述语言，因此提供了 AtomicProcesses、CompositeProcesses 和 ExcutingProcesses 三个执行进程。这样，一个执行引擎能调用 AtomicProcesses 去转化 WSDL；CompositeProcesses 使用 Sequence, Unordered 和 Split 控制结构；ExcutingProcesses 譬如依赖 If-Then-else 和 RepeatUntil 条件，而不支持默认执行，但能支持一种自定义语法和评价处理的解决方法；同时，还提供了转化处理的数据结构。但 OWL-S ontology 为每个服务提供了多个 Profiles，而 OWL-S API 只每个服务提供了一个 Profile，且 OWL-S API 不支持 ConditionalOutput 和 ConditionalEffect。

WSDL 注释程序集是一种基于机器学习的半自动的语义 Web 服务注释工具，其采用 point-and-click 接口向每一个 Web 服务注释语义，特征是建议在 WSDL 中用本体分类去实现每个 Web 服务的注释，而这个分类就是基于机器学习算法来实现，目的在于实现 Web 服务自动

地发现服务、组合服务和松散耦合调用软件组件。最终有效实现 WSDL 向 OWL-S 转化。

基于 WSDL2OWLS 的 OWL-S 可视操作框架是在 WSDL2OWLS 程序集上提供的一种可视化操作框架，因此主要提供了一个通用性的 OWL-S 描述框架，创建 involves OwlsWiz 去实现 WSDL 与 OWL-S 间映射，采用可视化操作实现转化，用 DRS 和 SWRL 的子集去简化逻辑表达表示方法，数据流结构用一个标记绑定机制实现生成，采用 Jena 实现逻辑推理，用户使用 GraphViz 去描述 RDF triples 的有向图。

3.4.2.2 OWL-DL 与 TMDM

TMDM 是通过 ISO 13250-2 发布的，如图 3-76 所示（在图中描述了主题图数据模型联系）；是主题图数据模型标准，用于定义 XTM 和作为 Topic Map query language (TMQL)①和 the Constraint language (TMCL)②的数据基础，而在本处内容主要讨论 OWL-DL 与 TMDM 的映射关系，以达到 OWL 与 TMDM 融合；但 ZHAO Tian-zhong 等人指出 RDF 与主题图具有相同的中心概念[9]，图 3-77 所示是 OWL 与主题图标准对比关系，从图中可知，主题图和 OWL 是两个不同的标准组合提出的一种语义表达方式，但它们都是建立在信息和资源间的元级数据模型描述关系；并且在 TMDM 标准中明确了与 OWL-DL 的等价转换关系，因此主要表现在：

（1）允许用户通过 TMDM 去创建 OWL-DL 主题图；

（2）允许用户使用 OWL 的特征去支持 TMDM 的语义，并且允许用户增加额外的语义去约束 OWL-DL；

（3）使得主题图有一个描述逻辑的语义。如表 3-22 所示是主题图与 OWL 术语的对照。

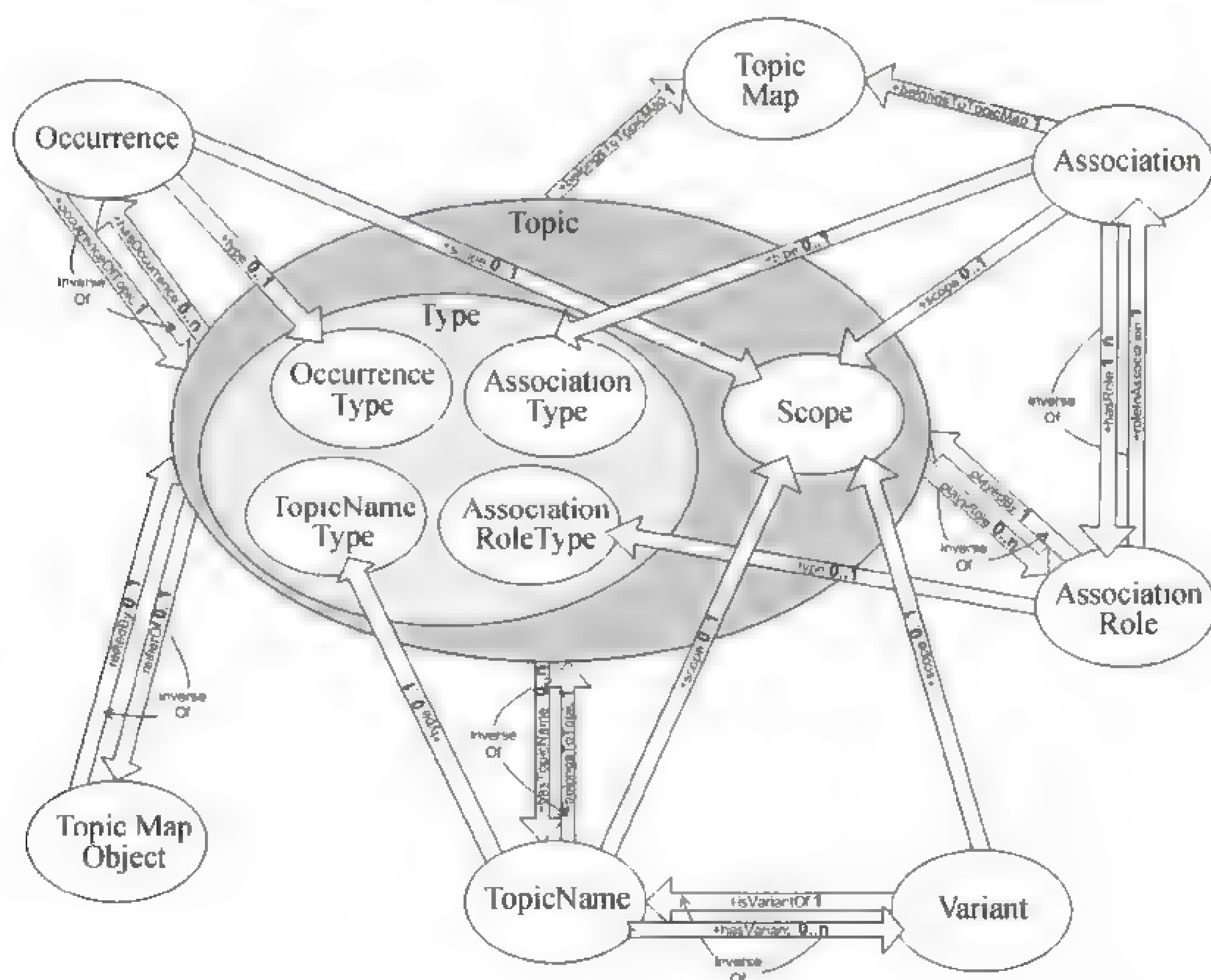

图 3-76 TMDM 结构图

① http://www.isotopicmaps.org/tmql/

② http://www.isotopicmaps.org/tmcl/

ISO Strandard			W3C Strandard
TMQL	TMCL	Constraints	OWL
Topic Maps		Data models	RDFS
			RDF
XTM		Syntaxes	XML/RDF

图 3-77　OWL 与主题图标准的对照

表 3-22　主题图与 OWL 术语映射表

Topic map term	Relationship	RDF term
Topic map	comparable to	RDF graph
Topic	comparable to	Resource
Subject	comparable to	Resource
Resource	comparable to	Network-retrievable resource
Non-addressable subject	comparable to	Non-network-retrievable resource
Association	kind of	Statement
Occurrence	kind of	Statement
Name assignment	type of	Statement
Class of topics	comparable to	Class

这就充分证明 OWL-DL 与主题图的转换是可行的，而且是可用的。在 TMDM 中给出了转换映射的描述方法，如下是 TMDM 标准提供的 OWL 类为 TMDM 定义的实体：

```
OWL classes are created for each TMDM Entity
<owl:Class rdf:id="Topic"> <owl:Class rdf:id="Occurrence">
<owl:Class rdf:id="TopicName"> <owl:Class rdf:id="Variant">
<owl:Class rdf:id="Association">
<owl:Class rdf:id="Association Role">

<owl:Class rdf:id="Scope">
<rdfs:subclassOf rdf:resource="#Topic"/>
</owl: Class>
<owl:Class rdf:id="Type">
<rdfs:subclassOf rdf:resource="#Topic"/>
</owl: Class>
<owl:Class rdf:id="AssociationType">
<rdfs:subclassOf rdf:resource="#Type"/>
</owl: Class>
<owl:Class rdf:id="AssociationRoleType">
<rdfs:subclassOf rdf:resource="#Type"/>
</owl: Class>
<owl:Class rdf:id "OccurrenceType">
<rdfs:subclassOf rdf:resource-"#Type"/>
</owl: Class>
<owl:Class rdf:id "TopicNameType">
```

```
<rdfs:subclassOf rdf:resource "#Type"/>
</owl: Class>

baseLocator:
<owl:Class rdf:id="TopicMap"/>
<owl: DataProperty rdf:ID= "baseLocator"
<rdf:domain rdf:resource="#TopicMap"/>
<rdfs:range rdf:datatype= "&xmls;string"/>
</owl:DataProperty>
......
```

现在面向服务化的语义软件方法通过以文献 Nikita Ogievetsky 提出的方法、The Markup Conference 2008[①]和 TMDM 标准进行分析主题图与 OWL-DL 转化过程，特别在 The Markup Conference 2008 中以都柏林核心数据为例实现了主题图与 RDF 的转换，介绍了 RDF 向 Topic Maps 的转换，Topic Maps 向 RDF 转换。因此可以将 Amazon 中的订购商品的 RDF 进行转换，并借助 TM4L[②]的主题图外挂编辑环境进化编辑和转换，如下是 RDF 向 Topic Maps 转换的列表示例：

```
<rdf:Description rdf:about=" http://amazon.com/">
         <amazon:title> Amazon - Home Page</ amazon:title>
         <amazon:creator> Amazon </amazon:creator>
         </rdf:Description>
      <topic id=" ">
         <subjectIdentity>
             <subjectIndicatorRef xlink:href=" http://amazon.com " />
         </subjectIdentity>
         <baseName>
              <baseNameString> Amazon - Home
                Page</baseNameString>
         </baseName>
         <occurrence>
             <instanceOf>
                <topicRef xlink:href="# " />
             </instanceOf>
           </occurrence>
      </topic>
      <topic id=" ">
         <baseName>
            <baseNameString> Amazon </baseNameString>
         </baseName>
      </topic>
      <topic id=" ">
        <baseName>
           <baseNameString> Creator </baseNameString>
        </baseName>
```

① http://www.balisage.net/Proceedings/vol1/html/Dichev01/BalisageVol1-Dichev01.html

② http://compsci.wssu.edu/iis/nsdl/download.html

```
        <baseName>
            <scope>
                <topicRef xlink:href="# " />
            </scope>
            <baseNameString>Resource</baseNameString>
         </baseName>
         <baseName>
           <scope>
              <topicRef xlink:href="# " />
           </scope>
           <baseNameString>Value</baseNameString>
        </baseName>
      </topic>
      <association>
         <instanceOf>
            <topicRef xlink:href="# " />
         </instanceOf>
          <member>
              <roleSpec>
                  <topicRef xlink:href="# " />
               </roleSpec>
               <topicRef xlink:href="# " />
          </member>
          <member>
               <roleSpec>
                   <topicRef xlink:href="# " />
               </roleSpec>
               <topicRef xlink:href="# " />
          </member>
      </association>
      <topic id=" ">
        <baseName>

        </baseName>
      </topic>
```

3.4.3　OWL 与 Web 服务、主题图的关系

在上面的相关小节中通过从技术和方法上分析了 OWL、Web 服务（WSDL）和 Topic Maps 间的转换是可行的，即要满足如图 3-78 所示的结构。从图中可以表明，Web 服务是不能直接与 Topic Maps 进行转换的，也就是 Web 服务不能直接访问主题图，需要通过中间媒介 OWL 去连接，也回答了最初所提及的 Web 服务位于主题图上层，但通过语义对 Web 服务的描述，即语义 Web 服务是可以直接访问主题图的；但在现实应用中，往往是很难访问到具体主题的，也很达到服务组合导航定位，即不能直接使用主题图的特征，造成主要原因有：

（1）语义 Web 与主题图采用的标准不一致，语义 Web 采用的 W3C 标准，主题图采用的

是 ISO 标准，如图 3-77 所示，这就使得它们的语法、语义结构和描述方法存在差异。

（2）语义 Web 与主题图主旨不一样，语义 Web 侧重表达语义识别、自动推理，主题图侧重知识管理，即对资源管理和组织。

（3）语义 Web 服务与主题图的 TMDM 存在不确定性，语义 Web 服务是松散耦合的结构，访问服务功能是通过 WSDL 去实现的，而主题图实际是一种语义网，而 WSDL 缺乏语义描述。因此，Web 服务与主题图间的访问必须借助 OWL 来实现，借助 OWL 来实现服务组合导航定位，借助 OWL 实现 Web 服务访问主题图。

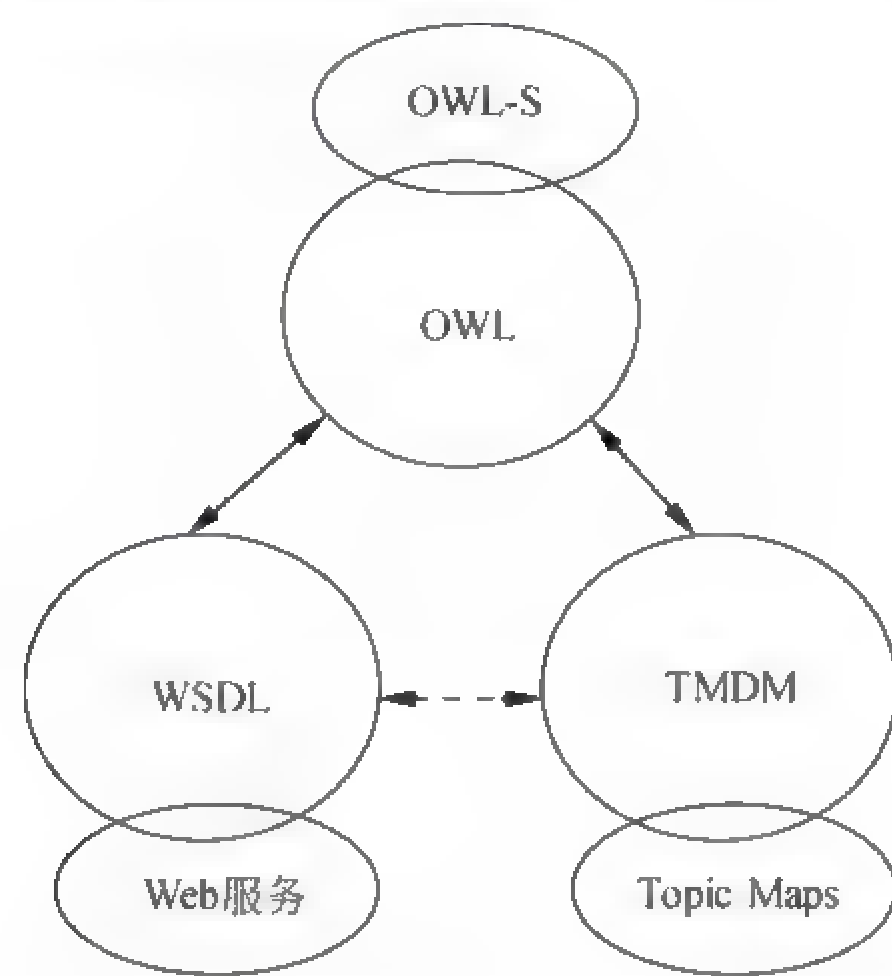

图 3-78　语义 Web 服务与 Topic Maps 间的融合图

3.4.4　面向服务软件语义化基础

计算机里的本体由概念、属性、关系、功能、实例和公理，以及事件来描述、抽象客观存在的事物，并用关系、公理来具体实现语义。它的描述基础就是 OWL-DL（Ontology Web Language Description Logic），并允许相关基元随意地与 RDF 与 RDFS 相互结合。面向服务化的软件语义化就是以描述逻辑为基础的本体来实现 WSDL 语义化，以及语义 Web 服务与主题图融合。

定义 3-1　本体（Ontology）是一个六元组 $Ont=<C,A^C,R,A^R,H^C,X>$，其中 C 表示概念集，A^C 表示每个概念的属性集，R 表示关系集，A^R 表示每个关系的属性集，H^C 表示概念层次，X 表示对本体的约束公理集。则一个本体集合表示为 $ont=\{x|Ont, x$ 表示本体集合中的个数$\}$。

定义 3-2　本体描述（Ontology Description）是一个五元组 $OD=<A^C(c_i), r_i(c_p, c_q), A^R(r_i), H^C\subset C_p\times C_q, X^{Ont}>$，其中 $A^C(c_i)$是概念属性集描述，$r_i(c_p, c_q)$表示概念间的关系，$A^R(r_i)$表示关系的属性，$H^C\subset C_p\times C_q$ 表示概念层次关系，X^{Ont} 表示公理间的约束关系。

定义 3-3　本体关系（Ontology Relation）是一个四元组 $OR=<Pof,Kof,Aof,Iof>$，分别表示 part-of、kind-of、attribute-of、instance-of 概念关系。其 OWL 基本建模元素的语义解释如表 3-23、表 3-24 所示。

表 3-23 OWL 基本建模元素：类构造算子与 DL 的关系

类构造算子	DL 语法	FOL(一阶逻辑) Syntax
IntersectionOf	$C_1\cap\ldots\cap C_n$	$C_1(x)\wedge C_2(x)\wedge\ldots\wedge C_n(x)$
unionOf	$C_1\cup\ldots\cup C_n$	$C_1(x)\vee C_2(x)\vee\ldots\vee C_n(x)$
complementOf	$\neg C$	$\neg C(x)$
oneOf	$\{x_1,\ldots,x_n\}$	$x=x_1\vee\ldots\vee x=x_n$
allValuesFrom	$\forall P.C$	$\forall y.P(x,y)\rightarrow C(y)$
someValuesFrom	$\exists P.C$	$\exists y.P(x,y)\wedge C(y)$
value	$\exists P.\{x\}$	$\exists x.P(x)$
maxCardinality	$\leqslant nP.C$	$\exists^{\leqslant n}y.P(x,y)$
minCardinality	$\geqslant nP.C$	$\exists^{\geqslant n}y.P(x,y)$
cardinality	$=np.C$	$\exists^{=n}y.P(x,y)$

表 3-24 OWL 基本公理元素：类构造算子与 DL 的关系

类构造算子	DL 语法
subClassOf	$C_1\subseteq C_2$
equivalentClass	$C_1\equiv C_2$
disjointWith	$C_1\subseteq\neg C_2$
sameindividualAs	$\{x_1\}\equiv\{x_2\}$
differentFrom	$\{x_1\}\subseteq\neg\{x_2\}$
subPropertyOf	$P_1\subseteq P_2$
inverseFunctionalProperty	$T\subseteq\leqslant 1P^-$
equivalentProperty	$P_1\equiv P_2$
inverseOf	$P_1\equiv P_2^-$
transitiveProperty	$P^+\subseteq P$
Transitive	$P^+\subseteq P^-$
Symmetric	$P\equiv P^-$

若分别采用 $P(x,y)$、$K(x,y)$、$A(x,y)$，$I(x,y)$分别表示在 Ont 中的概论间的关系：part-of，kind-of，attribute-of，instance-of。

定义 3-3.1 关系 direct_contain(x,y)满足

$P(x,y)\rightarrow$ direct_contain(x,y)

$K(x,y)\rightarrow$ direct_contain(x,y)

$A(x,y)\rightarrow$ direct_contain(x,y)

$I(x,y)\rightarrow$ direct_contain(x,y)

定义 3-3.2 $O=\{x|x\in Ont\}$是本体的概念集。

定义 3-3.3 关系 contain(x,y)满足：

direct_contain$(x,y)\rightarrow$ contain(x,y)

contain$(x,z)\wedge$ contain$(z,y)\rightarrow$ contain(x,y)

定义 3-3.4 intersection(x,y)满足

intersection$(x,y)=\{z|$ contain$(x,z)\wedge$ contain$(z,y),z\in O\}$

定义 3-3.5 union(x,y)满足

union$(x,y)=\{z|$ contain$(x,z)\vee$ contain$(z,y),z\in O\}$

定义 3-3.6 difference(x,y)满足

difference$(x,y)=\{z|$ contain$(x,z)\wedge\neg$contain$(z,y),z\in O\}$

定义 3-3.7 本体 *Ont* 内代数是 $N=(O,R,Op)$，分别表示 O 是 *Ont* 上的概念集；R 是 O 中的概念之间的关系集合；Op 是对 O 中的概念的操作集合。

定义 3-3.8 若满足下列条件，则称$\mathfrak{I}=(O,R,Op)$是本体 *Ont* 内的基本代数：

（1）$\mathfrak{I}$是 *ont* 内的代数；

（2）(direct_contain,contain)$\subset R\subseteq(P,K,A,I,$direct_contain,comtain)；

（3）$Op=($ intersection, union, difference)。

定义 3-4 本体特性（Ontology Characteristic）是一个三元组 $OCC=\langle Re^{Ont},Sh^{Ont},Co^{OD}\rangle$，分别表示本体重用、共享和一致性认证。且满足 Re^{Ont}(Class)$\rightarrow OD(Ont)\times Co^{OD}$，$Sh^{Ont}$(Public)$\rightarrow Co^{OD}(Ont\times Ont)$，$Co^{OD}=\langle X^{Ont}{}_1,X^{Ont}{}_2,\cdots,X^{Ont}{}_n\rangle$。

定义 3-5 本体分类（Ontology Classification）是一个三元组 $OC=\langle So,To,Co(So,To)\rangle$，分别表示源本体、目标本体和候选本体。

本体分类 DL 表达如下：

$\forall O_1,O_2\in Ont,\exists oc\rightarrow OC$

if $O_1\wedge O_2\rightarrow\mathfrak{I}$, then $\exists oc(OCC)\rightarrow OC.(So\cap To)\neq\varnothing$

if $O_1\wedge O_2=\varnothing,\exists\neg O_1\wedge O_2\rightarrow\mathfrak{I}$ or $\exists O_1\wedge\neg O_2\rightarrow\mathfrak{I}$, then $\exists OC.(Co(So,To))\rightarrow OC.(So\cap To)\neq\varnothing$

if $O_1\vee O_2\rightarrow\mathfrak{I}$, then $O_1\equiv O_2$

if $O_1\vee O_2=\varnothing$, then $O_1=\varnothing$ and $O_2=\varnothing$

else if $\exists\neg O_1\vee O_2\neq\varnothing$, $O_1\vee\neg O_2\neq\varnothing$

and $\neg O_1\wedge\neg O_2\rightarrow\mathfrak{I}$, then $OC.So\wedge OC.To\neq\varnothing$

定义 3-6 本体映射（Ontology Mapping）是一个四元组 $OM=\langle Sim,Mp,Sim,Matc\rangle$，分别表示本体的相似度、映射模式、相似度和匹配度。

其中本体的相似度计算采用微观的定量方法和余弦相似相结合的方法，这就避免了本体相似计算的单一性，如式（1）。

$$Sim(Ont_i,Ont_j)=\frac{|Co(So,To)|}{|So|_i+|To|_j}\frac{\sum_{u\in U}O_{u,i}O_{u,j}}{\sqrt{\sum_{u\in U}O_{u,i}{}^2\sum_{u\in U}O_{u,j}{}^2}} \quad (1)$$

式中| |表示取数，(u,i)、(u,j)分别表示来自候选集合的本体——源本体、目标本体集合向量，U 表示候选本体集合中的本体。

定义映射模式集合 $Mp=\{mp_1,mp_2,\ldots,mp_n\}$，且为每种模式的分配定义一个方阵 R：

$$\boldsymbol{R_{ij}}\begin{bmatrix} r_{11} & r_{21} & \cdots & r_{n1} \\ r_{21} & r_{22} & \cdots & r_{n2} \\ \vdots & \vdots & & \vdots \\ r_{n1} & r_{2n} & \cdots & r_{nn} \end{bmatrix}$$

，求得 R 的一组特征值$\xi=[\lambda_1,\lambda_2,\ldots\lambda_r]$，并得一个特征向量 $P_{u,j}$，

且使得$\sum_{i=1}^{t}\lambda_i w_i = 1$，$w$ 是分配给不同特征值的权重，并针对本体相似选择状态对本体匹配进行预测，如（2）。

$$PV_{u,i} = \frac{\sum_{j\in U} Sim(Ont_i, Ont_j) \times P_{u,j}}{\sum_{j\in U} |Sim(Ont_i, Ont_j)|} \tag{2}$$

这时，可以得到本体的匹配度，如式（3）。

$$Match(Ont_i, Ont_j) = \max[Sim(Ont_i, Ont_j)] \times PV_{u,i} \times \lambda_i w_i \tag{3}$$

定义 3-7　本体实例（Ontology Instance）是一个三元组 $OI=<Md,Je,Ra>$，分别表示实现本体推理的元数据、推理分析器和推理器。

定义 3-8　本体模式（Ontology Mode）定义为 $OMe=<Ont,OD,OR,OCC,OC,OM,OI>$。

定义 3-9[10]　主题图（Topic Map）是一个七元组：$TM=<T_C,T_O,T_A,T_R,T_I,R_H,R_A>$，其中：$T_C$ 表示主题类型集合；T_O 表示事件（资源出处）类型集合；T_A 表示关联类型集合；T_R 表示角色类型集合；T_I 表示一个实例主题集合；R_H 表示层次关系集合；R_A 表示关联关系集合；且相同的 T_C 扮演不同的 T_R。

定义 3-10　主题图执行（Topic Maps Execution）是一个三元组：$TAO=<TS,AR,RO>$，其中：TS 表示以主体（Subject）来表达主题（Topic）；AR 表示以关系（Relationship）来表达关联（Association）；RO 表示事件（Occurrence）与主题的连接。

定义 3-11　知识库（Knowledge Base）是一个五元组 $KB=<C_{KB},R_{KB},I,l_C,l_R>$，其中集合 I 的元素是实例标识符，函数 $l_C:C_{KB}\rightarrow\beta(I)$是本体概念实例化，若对于所有的 $r\in R$，有 $l_R(r)\in\prod_{c\in\sigma(r)} l_C(c)$，则 $l_R:R_{KB}\rightarrow\beta(I^r)$本体关系实例化。

定义 3-12　描述逻辑知识库[11]是一个二元组 $DLKB=<KB.TBox, KB.Abox>$，其中 $TBox$ 描述模型域的结构，$ABox$ 描述具体的形态，如数据和事实等，而语法结构如表 3-25 所示。

表 3-25　DLBK 的语法结构

Concepts	Description	Expressions
Atomic	A,B	For A, Subset of Δ^I
Not	¬C	$\Delta^I \backslash C^I$
And	C∩D	{x| x∈C^I and x∈D^I}
Or	C∪D	{x| x∈C^I or x∈D^I}
Exists	∃R.C	{x|(x,y) ∈R^I and y∈C^I }
For all	∀R.C	{x|if (x,y) ∈R^I then y∈C^I}
At least	⩾n R.C(⩾n R)	{x|#{(x,y) ∈R^I and y∈C^I }⩾n}
At most	⩽n R.C(⩽n R)	{x|#{(x,y) ∈R^I and y∈C^I }⩽n}
Nominal	$\{i_1,i_2,\ldots i_n\}$	$\{i_1^I,i_2^I,\ldots i_n^I\}$
Role	Description	Expressions
Atomic	R	Subset of Δ×Δ
Inverse	R^-	{(y,x)|(x,y)∈R^I}
Concept Axioms(TBox)	Description	Expressions
Subclass	C⊆D	$C^I \subseteq D^I$
Equivalent	C≡D	$C^I \equiv D^I$

续表

Concepts	Description	Expressions
Role Axioms(RBox)	Description	Expressions
Subrole	$R \subset S$	$R^I \subset S^I$
Transitivity	Rrans(S)	Rrans(S^I)
Assertional Axioms	Description	Expressions
Instance	C(a)	$a^I \in C^I$
Role	R(a,b)	$(a^I, b^I) \in R^I$
Same	a=b	$a^I = b^I$
Different	$a \neq b$	$a^I \neq b^I$

定义 3-13 资源状态（Resources State）是一个四元组 $RS=<owl\text{-}s, rdf, xtm, link(url, urn, uri)>$，分别表示功能描述，本体描述，资源描述，主题图描述和资源连接方式，该元组的项数可以根据具体情况增加其他类型的描述语言。它与描述逻辑知识库的关系表达为：$RS \rightarrow DLKB$。

定义 3-14 主题图知识库表示是一个五元组 $TMKB=<TM, TAO, RS, Cr, Cc(sd, cf)>$，其中 Cr 表示识别器，$Cc(sd,cf)$ 表示对知识分类，sd,cf 分别表示分类决策和分类器，且满足 $<RS, Cr(TM) \rightarrow Cc(TM)> \rightarrow TAO$，则表示对 $TMKB$ 实例化。

3.4.5 面向服务的软件语义化方法

由于 OWL DL 与 SHOIQ 是等价的[8]，因此采用 SHOIQ 实现语义 Web 服务访问主题图，其描述逻辑推理采用 Tableau 算法来实现，它能够解决描述逻辑概念的可满足性问题和推理。Ian Horrock 等人提出了一种针对 SHOIQ 的 Tableau 决策法的方法，详细介绍了 Tableau 算法在 SHOIQ 中推理方法和可满足性判定，并通过定义 SHOIQ 知识库来表达。而描述逻辑是一阶逻辑（FOL），因此，就需要将 SHOIQ 映射到 FOL，映射规则如表 3-26 所示，映射结果如表 3-27 所示。这时，语义 Web 服务只要通过 SHOIQ 就能进行相互映射主题图。

表 3-26 Negation Normal Form(NNF) and Transformation rules(TR)

Rewrite rules	Transformation rules(tr)		
$\neg(C \cap D) \rightarrow \neg C \cup \neg D$	$\{(C_1 \cap C_2)(x), \ldots\}$	$\rightarrow \cap$	$\{(C_1 \cap C_2)(x), C_1(x), C_2(x) \ldots\}$
$\neg(C \cup D) \rightarrow \neg C \cap \neg D$	$\{(C_1 \cup C_2)(x), \ldots\}$	$\rightarrow \cup$	$\{(C_1 \cup C_2)(x), C_1(x), \ldots\}; \{(C_1 \cup C_2)(x), C_2(x), \ldots\}$
$\neg \forall R.C \rightarrow \exists R. \neg C$	$\{(\exists R.C)(x), \ldots\}$	$\rightarrow \exists$	$\{(\exists R.C)(x), R(x,y), C(y) \ldots\}$
$\neg \exists R.C \rightarrow \forall R. \neg C$	$\{(\forall R.C)(x), R(x,y), \ldots\}$	$\rightarrow \forall$	$\{(\forall R.C)(x), R(x,y), C(y) \ldots\}$

表 3-27 SHOIQ 映射 FOL 的结果

A(atomic concept)	A(x)	a∈A	A(a)
$\top$	$\top$	$<a,b> \in R$	$R(a,b)$
$\bot$	$\bot$	$C \subseteq D$	$\forall x: tr(C,x) \rightarrow tr(D,x)$
$C \cap D$	$tr(C) \wedge tr(D)$	$C \equiv D$	$\forall x: tr(C,x) \leftrightarrow tr(D,x)$
$C \cup D$	$tr(C) \vee tr(D)$	$Q \subset R$	$\forall x,y: Q(r,y) \rightarrow R(x,y)$
$\neg C$	$\neg tr(C)$	$R \equiv Q^-$	$\forall x,y: R(x,y) \leftrightarrow R(y,x)$
$\forall R.C$	$\forall y: R(x,y) \rightarrow tr(C,y)$	$R^+ \subset R$	$\forall x,y,z: R(x,y) \wedge R(y,z) \rightarrow R(x,z)$

续表

A(atomic concept)	A(x)	a∈A	A(a)
$\exists R.C$	$\exists y: R(x,y) \wedge tr(C,y)$		
$\{o_1,\ldots,o_n\}$	$x=o_1 \vee \ldots \vee x=o_n$		
$\exists R.\{o\}$	$R(x,o)$		
$\geqslant nR$	$\exists y_1,\ldots,y_n: \wedge R(X,y_i) \wedge \wedge y_i \neq y_j$		
$\leqslant nR$	$\forall y_1,\ldots,y_{n+1}: \wedge R(X,y_i) \rightarrow y_i \neq y_j$		

3.4.5.1　基于 Tableau 的语义 Web 主题服务组合方法

语义 Web 主题服务组合（SWTSC）与单纯的语义 Web 服务组合（SWSC），有着一个重要的区别：SWTSC 具备了主题图的特征和性质，同时也具备现有 SWSC 的概念、特征和性质；因此在服务组合上也有所不同。在本书并采用 SHOIQ 来实现 SWTSC 的描述逻辑实现，先定义 SWTSC 如下：

定义 3-15　SWTSC 是一个七元组合 $SWTSC=\langle RP,RQ,UDDI,SD.WSDL,QoS,CF,WSCR\rangle$，其中：

（1）RP 表示请求语义 Web 服务主题，可表示 $RP=\{(rp_1,rp_2\ldots,rp_n),\rightarrow\}$；

（2）RQ 表示响应语义 Web 服务主题，可表示 $RQ=\{(rq_1,rq_2\ldots,rq_m),\leftarrow\}$；

（3）*UDDI* 是 SWTSC 注册中心，并与语义有着紧密的联系；

（4）*SD.WSDL* 表示基于 OWL-S（或 WSMO）的服务功能描述，并能满足逻辑描述规则；

（5）*QoS* 是 SWTSC 的服务质量，一般由时间，花费，可用性、安全性和可访问性等组成；

（6）*CF* SWTSC 流程，一般可以包括串行，并行等；

（7）*WSCR* 表示语义 Web 主题服务关系，如整体-部分等关系。

根据 Horrocks, I 等人所提出的方法[11]，在 SHOIQ 中的所有概念 C 都遵守 Negation Normal Form(NNF) and Transformation rules(TR)，因此，针对一个概念 C，采用 $nnf(C)$ 去表示 NNF，用 $tr(C)$ 表示 TR，用 $Sub(C)$ 表示 C 的子概念，且 $Sub(C)\subseteq C$；对满足知识库 DLKB 的 $(SWTSC,R)$，定义“relevant subconcepts” $cl(SWTSC,R)$ 并根据参考文献[10]定义如（1）；定义 $tr(C)$ 的 TR 约束为（2）。

（1）
$$cl(SWTSC,R):=\bigcup_{C\subseteq D\in SWTSC} cl(nnf(\neg C\cup D),R)$$

式中

$$cl(E,R):=sub(E)\cup\{\neg C\mid C\in sub(E)\}\cup\{\forall S.C\mid \forall R.C\in sub(E)\ or\ \neg\forall R.C\in sub(E)\ \text{and}\ S\ occurs\ in\ SWTSC\ or\ R\}$$

式中，R 是一个转换角色名称，且 $E\subseteq SWTSC$。

（2）$\forall cl(SWTSC,R),\exists R(C,D)\rightarrow tr(SWTCS,\pi),\pi=\{\cup,\cap,\otimes,\oplus\}$

式中

$$tr(ST,\pi):=\{\exists R.C\mid ST\ \pi\ C\cap ST\ \pi\ D\neq\varnothing\ \text{and}\ C\cap D=\varnothing\}\cup\{\neg ST\mid ST\in SWTCS\}$$

定义 3-16　$(WSTSC,R)$是一个基于 $DLKB$ 的知识库，则 Tableau T 为$(SWTSC,R)$可以定义为：$T\ (S,L,\varepsilon)$，S 表示 $SWTSC$ 的个体集合，且 $L:S\rightarrow 2^{cl(SWTSC)}$，$\varepsilon:tr(C)\rightarrow 2^{S\times S}$，这时，对于$\forall s,t\in S$,

$C,C_1,C_2 \in cl(WSTSC)$；且 $R,tR \in tr(C)$，这时根据参考文献[10]有：

```
if C⊑D∈ SWTSC, then nnf(¬C⊔D)∈L(s).
if <s,t>→ ε(WSTSC), then tr(s,t)∈ ε(R).
if C∈L(s),then ¬C∉ L(s).
if tr(C)∈ ε(R),then ¬tr(C) ∉ε(R).
if C1⊓C2∈ L(s), then C1∈ L(s) and C2∈ L(s).
if C1⊔C2∈ L(s), then C1∈ L(s) or C2∈ L(s).
if ∀R.C∈ L(s) and <s,t>∈ε(R), then C∈ L(s).
if ∃ R.C∈ L(s) and s∈S,then <s,t>∈ε(R) and C∈L(t).
if ∀S.C∈ L(s) and <s,t>∈ ε(R)and tr(R,tR)⊑tr(ST,π), then ∀R.C∈L(t).
if (≥nS.C) ∈ L(s),then ∃{t∈S,R∈tr(C)| <s,t>∈ε(R) and C∈L(t)} ≥n.
if (≤nS.C) ∈ L(s),then ∃{t∈S,R∈tr(C)| <s,t>∈ε(R) and C∈L(t)} ≤n.
if o∈ L(s)⊓ L(t) and <s,t>→iff o, then s=t.
```

定义 3-17 $(WSTSC,R)$的 Tableau 决策过程是一个完全有向图 $G=(V,E,L,\neq)$，则对于每个结点 $x\in V$ 可以定义集合[10]：

$$L(x)\subseteq cl(T)\cup\{(\leqslant mR.C)\mid(\leqslant nR.C)\in cl(T)\ \text{and}\ m\leqslant n\}$$

每一条边 $e=<x,y>\in E$ 是 $L(e)$规则的名称，$\neq$是两个结点间的判断符号。这时，增减结点和边的操作如下：

$Add(C,L(x))$: $L(x)\cup\{C\}\rightarrow L(x)$ or $\{C\}\cup L(y)\rightarrow L(y)$ and $\{<x,C>\}\in L(e)\cap\{<C,y>\}\in L(e)=\{C\}$.

$Del(L(x),C)$: $L(<x,y>,C)\cap\{C\}=\varnothing$ and $<x,y>\in L(e)$.

$Add(<p,q>,L(<x,y>))$: $L(<x,y>)\cap\{<p,q>\}\rightarrow L(<x,y>)\cup L(<p,q>)$ and $L(y,p)\in L(e)$.

$Del(L(<x,y>,L(<p,q>))$: $L(<p,q>)\cap\{\forall L(e)\}=\varnothing$,and $(<p,q>)\notin G$.

$x=y$ 表示结点 x,y 融合，详情见参考文献[47]。

$x\neq y$ 表示添加一个不等关系的结点 x,y: $\{(x,y)\}\cup\neq\rightarrow\neq$.

$Add(<p,q>,L(<x,y>))$: $L(<x,y>)\cap\{<p,q>\}\rightarrow L(<x,y>)\cup L(<p,q>)$ and $L(y,p)\notin L(e)$ and $(<p,q>)\in G$.

引理[10] 一个 SHOIQ 知识库（$WSTSC,R$）存在，当且仅当存在一个为（$WSTSC,R$）的 Tableau。

定理 3-1 一个语义 Web 主题服务组合满足 SHOIQ 描述，当且仅当存在一个满足$(WSTSC,R)$的 Tableau。

证明：由定义 3-16、定义 3-17 知，*WSTSC* 满足 NNF 和 TR 的性质，而由引理也知，一个 *WSTSC* 是满足 SHOIQ 描述的。

推论 语义 Web 主题服务组合（SWTSC）的过程由 Tableau 决策（由定理 1 易知）。

定理 3-2 若 WSDL 满足 OWL 描述以及 TMDM 满足 OWL 描述，则 WSDL 通过 SHOIQ 描述能访问 TMDM。

证明：由前几节可知，以 OWL 为中间描述对象，可以实现 WSDL 与 TMDM 的语法及语义判断与识别，又由定理 3-1 和推论可知，WSDL 可以访问 TMDM，即语义 Web 服务具备主题图的特征。

这时，根据语义 Web 主题服务组合的要求改进 Tableau 算法如下：

```
unfold: if A ∈L(x), A atomic and (A ⊂ D)∈ (SWTSC,R), then
(T (A,x)⊕{{ A ⊂ D }})∪ T (D,x)→ T (D,x)
```

```
if D∉L(x),then Add(D,L(x))
else Del(L(x),D)

⊏-rule: if C⊏ D∈ L(x), and nnf(¬C∪D)∉L(x), and (C π D)∈ ε(R)
then set L(x)=L(x) ∪{nnf(¬C∪D)} and meet L(x)←ε(R)

∩-rule: if C1∩C2∈L(x) and {C1,C2}⊄ L(x) and C1∩C2∈ε(R)
then set L(x)=L(x)∪{C1,C2} and meet L(x)← ε(R)
and Add({C1,C2},L(x))

∪-rule: if if C1∪C2∈L(x) and {C1,C2}∩ L(x)=∅ and {C1,C2}∩ L(x)∉ ε(R)
then set L(x)=L(x)∪{C} for some C∈{C1,C2} and meet L(x)∪{C}≠∅← ε(R)
and Add({C1,C2},L(x))

∃-rule: if ∃S.C∈L(x), x is not blocked, and x has no safe S-neighbor y with
C∈ L(x)
then create a new node y with L(<x,y>)={S} and L(y)={C} and L(<x,y>)∈ε(R)
and Add(C,L(y))

∀-rule: ∀S.C∈L(x) ∈ε(R), and there is an S-neighbor y of x with C∉L(y)
then set L(y)=L(y)∪{C} and meet L(y) ∈ε(R)
and Add(C,L(y))

∀+-rule: if ∀S.C∈L(x) ∈ε(R), and there is some R whit tr(R) and R,S∈ε(R),
and there is an R-neighor y of x with ∀R.C∉L(y) and L(y) ∉ ε(R)
then set L(y)=L(y)∪{∀R.C}
and Add(∀S.C, L(y))

≥-rule: if (≥nS.C) ∈L(x) ∈ε(R), x is not idnirectly blocked,
((S,<xi,yi>)∪((∀≥nS.C),x) →(S,<xi,yi>), for 1≤i≤n
and there are not n safe S-neighbor y1,…,yn of x with
C∈L(yi) ∈ε(R) and yi≠yj for 1≤i<j≤n,
then create n new nodes y1,…,yn with L(<x,yi>)={S}
C∈L(yi)={C} and yi≠yj for 1≤i<j≤n
Add (C,L(<xi,yi >))→L(e).

≤-rule: if(≤nS.C) ∈ L(x) ∈ε(R), x is not idnirectly blocked,
and there are m S-neighbor y1,…,ym of x with m>n, then
select two S-neighbor of x:y,z, let C∈L(y)∩L(z) ∈ε(R) and not y≠z
Generate α=(x,→ ≤,≤nS.C),{{Sx,y,Sx,z},…}
do |Δ|α→Δ
set y=z
(y=z)∪(( ≤nS.C),x)⊕(S,<x,ym>)⊕{{α}}⊕{ C∈L(y)∩L(z) }→(y=z)
if z is a nominal code, Merge(y,z)
if else if y is a nominal node or ancestor of z, Merge(z ,y)
```

```
else Merge(y,z)

O: if for some {o}∈ S, there 2 nodes x,y with {o}∈ L(x)∩ L(y)
(x-y)∪{{o},y} π {{o},x}→(x-y)
if not x≠y,then Merge(x,y), and set x-y
```

语义 Web 主题服务组合（WSTSC）算法：

输入：语义 Web 服务和基于 OWL 描述的主题图：即语义 Web 主题服务

输出：一个 WSTSCS 结果

方法：

（1）∀ 一个请求语义 Web 主题服务，∃ 一个响应语义 Web 主题服务，并且寻找 UDDI；

（2）

```
    if(Web 服务与主题图的语义描述)
    启用描述逻辑，并检查 Web 服务与主题图语义描述情况；
      if(满足语法和语义结构)
        while(各描述逻辑的文档标签)
           进行相似性对比；
           进行匹配；
           实现语义 Web 到主题图的映射；
      else
          抛出检查结果，并告之用户出错的原因
    else
     抛出判断结果，并指出语义描述存在的问题。
```

（3）启用 SHOIQ 规则，并调用 Tableau 进行决策：

```
   if(Tableau 决策)
     while(通过 Tableau 逐句验证描述逻辑)
       通过 Tableau 决策语义 Web 主题服务，并通过 SOAP 传递；
   else
```

抛出错误，并返回第（2）步重新判断

（4）反馈结果，若满足需求，则转到第（5）步，否则返回第（2）步或第（3）步。

（5）退出。

3.4.5.2 结果分析

应用实例通过网上订购图书登录所涉及的相关服务进行描述逻辑举例，因此：

（1）编写主题语义 Web 服务的业务功能和逻辑的文档，即语义 Web 服务访问主题图的文档。格式如下：

```
[ owl:class rdf:ID="login";
  rdf:subject  :user;
  rdf:type login service:userop;
  rdfs:domain rdf:resource-"#BinqComposer";
  rdfs:range rdf:resource-"#Input";
  rdfs:lable "username";
  :compose.opera[
```

```
    rdf:object  :dataservice;
    rdf:type login service:checking;
    rdfs:lable  "username";]
 ]
 ...
 [process:compositeProcess rdf:ID="LS"   //Tableau decision[18]
 ...
 ]
```

（2）配置 Web.xml 等配置文件。

（3）采用 Java 实现 TSWS 算法。

（4）设置三个服务计数器和一个计时器，分别用来计算请求服务数（*RW*）、响应服务数（*RS*）和服务组合数（*WS*），以及登记完成服务组合所需时间（*T*）。

现以用户登录系统为例说明基于 Tableau 进行说明主题语义 Web 服务的流程，并假设与登录相关的服务是一个知识库：

Login(user,password)先检查 unfold，若所产生的属性不冲突，则 user∈usertable→Add(user, usertable)。否则记录 Del(user,password)。

若所产生的属性不冲突，则 Login(user,password)⊑usertable∈*L*(user) and *L*(password)。并记录 Login(user,password)。

若所产生的属性冲突，则 Login(user,password)⊓usertable(user,password) ∈*L*(user) and *L*(password) and ≠∅，则记录 Add(user,usertable)和 Add(password,usertable)。

检查结束，则 Login(user,password)→(Add(user,usertable), Add(password,usertable))进入商品订购系统。

……

这时若满足系统验证规则，则进行用户操作页面。否则继续检查，直到满足规则为止。

下面分析是基于主题语义 Web 服务与传统方法的服务组合数的比较，图 3-79 是分析服务组合结果示意图。这个服务结果是以登录服务、信誉服务、验证服务、区域设置服务、权限认识服务、查询服务、搜索服务、筛选服务、订购服务、提交服务和结算服务等基本服务组成，并指向不同的主题资源（Subject Resource），但在实际登录时还会有其他服务参与。

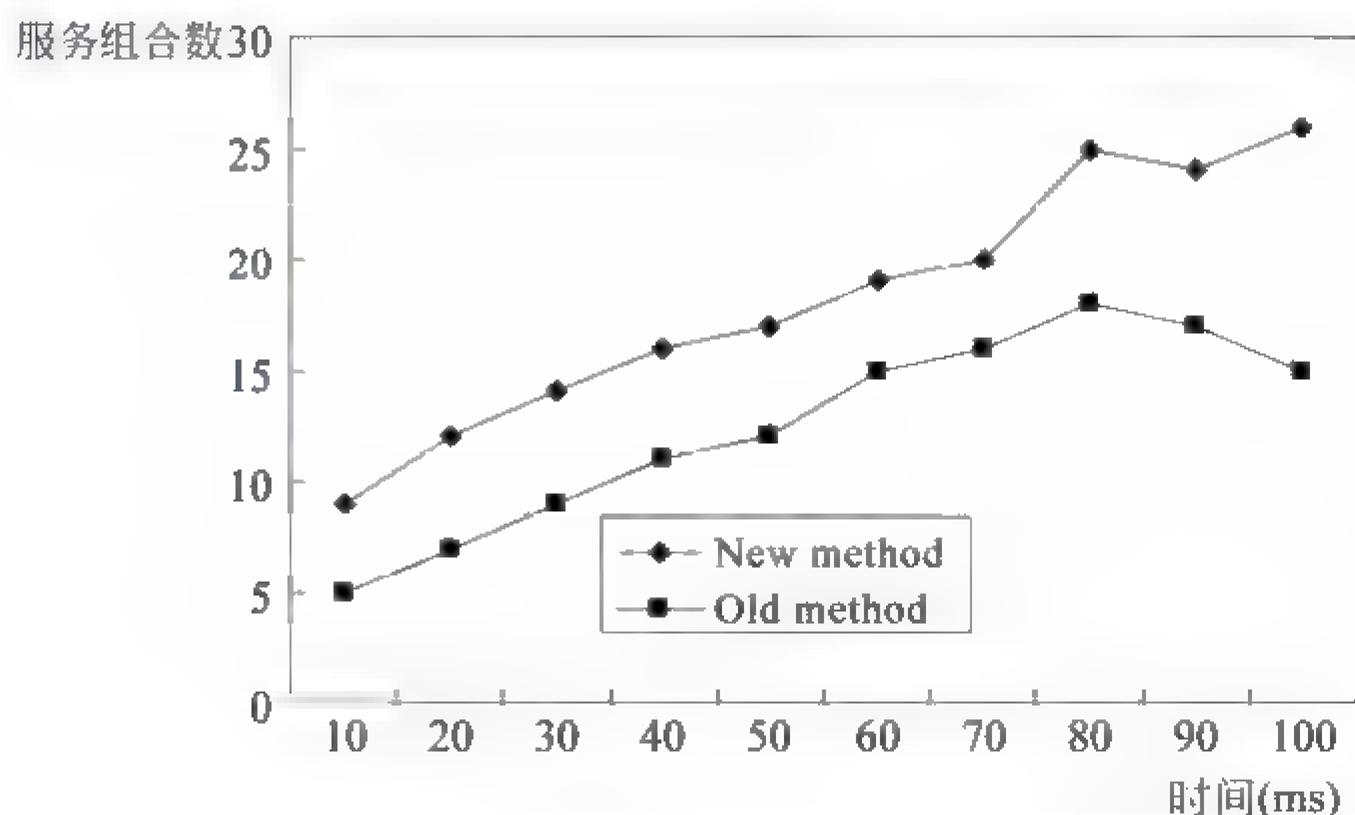

图 3-79　面向语义 Web 服务+主题图的方法与传统方法服务组合数比较示意图

3.4.6 面向服务的软件语义化研究进展

从服务、服务组合及主题图两角度阐述近来相关的工作和发展方向，其中服务和服务组合是近来分布式信息集成、计算最重要的方法之一，由于它具有分布性能强，跨平台、语言无关性好，部署灵活等优势。而主题图自 1999 标准问世以来，虽然经过三次对主题图标准的更改和丰富，以及最近对前三次主题图标准再一次发布，但研究范围仍存在局限性，应用领域不广，除了在图书馆知识管理和相关的应用外，其他应用领域还仍需要进一步探索。因此，本书提出一种语义 Web 服务与主题图融合方法，并以开源件建立语义 Web 服务与主题图的应用框架。

3.4.6.1 服务及服务组合相关工作

相关本体、Web 服务的研究已经在多个领域里得到了应用，而本体在 Web 服务中主要起到语义描述的作用，用于描述数据/信息，功能/操作、服务质量和执行等语义，但让服务直接去识别、请求和响应这些语义，又存在问题，这是因为 WSDL 描述语义不深，服务优先是通过 WSDL 去得到服务功能描述所造成的。因此，许多研究者从不同角度、不同领域、不同应用进行了研究，并提出了大量的有效的方法和应用的方法，大致可归结于形式化描述，智能计算、语义方法、逻辑推理描述、基于服务质量（QoS）的描述、常规描述方法和其他一些常规方法进行描述服务，其中语义方法是使用最多的方法之一，如表 3-28 所示；建立相关的服务组合理论来增强服务组合的通用性，其中有服务的代数结构方法[28]和线性逻辑的语义服务组合方法[29]；在对服务组合效率上主要以服务质量进行衡量，也有采用评价因子和多指标体系的多目标评价方法，如表 3-29 所示；在服务组合具体实现上，有以服务质量为基础的服务组合框架和面向服务体系结构的服务组合框架，如表 3-30 所示；而服务应用已经在电子政务[38]、地理信息系统[39]、电子招聘[40]等多个领域得到了应用。这从几个层次可以得到，目前面向服务计算的方法很多，研究的角度多，应用范畴广。

表 3-28 服务及服务组合方法

方法类别	描述	发表时间
形式化描述	邓水光等人建立了服务视图的π演算形式化服务行为兼容性定性判定方法[13]	2007.12
	王晓玲等人提出了基于文法的形式化处理服务接口信息处理方法[14]	2005.04
智能化计算	Wen-Yau Liang 等人提出了一种合并粗糙集的通用遗传算法去解决复杂和多系统的服务组合方法[15]	2009.03
	Jun Yana 等人和 John Debenham 等人采用了自治代理谈判技术和服水平协商去实现服务组合[16-17]	2007.02,2007.11
	Zakaria Maamar 等人提出了一种基于代理和面向上下文的服务组合方法[18]	2005.05
语义相关方法	Brahim Medjahed 等人采用了语法、静态语义、动态语义和定性分析四个层次建立一种多级语义服务可组合的方法[19]	2005.07
	KEITA FUJII 等人通过建立多种类型组件实现一种基于语义的动态服务组合体系结构[20]	2006.03
	Z. Maamar, N.C 等人提出了一种基于本体的上下文调和的方法去实现具有安全性和 specifying 的服务组合、描述整合结构[21]	2005.07

续表

方法类别	描述	发表时间
逻辑推理描述	国内著名学者史忠植等人提出了把语义 Web 服务组合问题转化为描述逻辑推理问题的方法[7]	2008.04
	Matteo Baldoni 等人首先聚焦在服务自动选择和组合上，然后从不同角度提出关于相互作用的协议推理方法去定制服务选择和组合[22,23]	2007,2004,2008
基于 QoS 描述	范小芹等人提出了 Web 服务各随机 QoS 指标的度量方法和自适应 QoS 管理体系结构去实现随机 QoS 感知的可靠 Web 服务组合[24]	2008.10
	Jong Myoung 等人提出了满足服务组合的约束 QoS 算法和服务组合规划体系结构[25]	2008,11
特定方法	Wen-Yau Liang 等人提出了 design with object (DwO)的服务组合方法，其采用面向对象的概念去实现[26]	2007
	Patrick N. Blessa 等人通过建立一种基于花费最小的服务映射方法去实现服务组合[27]	2006.10

表 3-29　服务及服务组合评估方法

评估方法类型	描述方法	发表时间
基于 QoS 的评估方法	Zeng I 等人采用了服务运行成本、运行时间、信任度、成功率和可用性来评价服务属性，评价者根据自身经验设定服务属性的权重，评价结果为这五个评价因子的取值加权后的总和来实现[30]	2004.05
	YANG Fang-chun 等人提出了一种基于不确定多属性决策理论的全局优化决策算法去实现混合 QoS 模型感知的语义 Web 服务组合，并以 QoS 去衡量服务组合情况[31]	2008.10
建立评估因子方法	杨文军等人提出了基于语义的方法描述 Web 服务评价模型，并支持动态定制不同领域的服务评价因子，同时利用 Web 服务在使用过程中产生的知识以及领域专家的领域知识交互计算评价因子的权重分布[32]	2005.04
多指标体系的评估方法	周相兵等人首先对服务进行了基于本体的语义描述方法，其次通过多服务所处的状态建立了多指标的评估体系和多指标计算的多目标方法，并通过改进遗传算法求解，然后通过 AHP 进行分析[33]。	2008.12

表 3-30　服务及服务组合框架方法

服务组合框架类型	描述	发表时间
基于 QoS 的框架方法	Gerardo Canfora 等人提出了一种面向 QoS 感知的绑定与再绑定的服务组合框架方法，并采用遗传算法去感知 QoS[34]	2008.01
基于形式化的框架方法	Yu-Liang Chi 等人分别介绍了一种形式化的、基于 Petri 流程建模型的服务组合框架[35]	2008
	KEITA FUJII 等人也提供了一种基于语义的动态构件方法去实现服务组合的体系结构[20]	2006.03
中间件方法	Shengwei Wang 等人提供了一种基于服务的集成和互操作的自动智能系统中间件[36]	2007
	Liangzhao Zeng 等人提出了一种 QoS 感知的服务组合中间方法[37]	2004.05

表 3-28 中各类型的方法的宗旨都是为了很好的描述服务，使其能有效完成服务组合，完成功能的需求，其中形式化描述的优点是具有严格的语法、逻辑和表达能力，但较抽象，难

于真正实现，用于实际的工作去。智能计算方法具有分布、并行和判断能力，能得到较好的结果，是最常用的方法之一，但对软件、硬件要求高，对新产生的服务难控制，在实现运用中有较好的表现。语义描述服务及服务组合能增强服务识别、抽取的能力，但语义不易操作，对服务的语义关联处理力度不大。逻辑推理描述可以有效解决智能计算和语义描述存在的突出问题，基于 QoS 描述的服务组合方法主要从花费、时间、可靠性和可用性等角度去衡量服务组合的情况，因此具有服务组合精确性、完整性等优点，但不易定量、难控制等；目前最常用的方法就是通过不同时刻建立矩阵，以矩阵值变化来量化。特定方法服务描述及服务组合是针对特定应用、特定的要求一种特殊的方法，这种方法也是最常用的方法之一。

表 3-29 中各类服务组合评估方法是当前采用最多评价方法，其中基于 QoS 的评估方法是应用范围最广，最有效的方法，优点在于对服务的特征进行了描述，得出了具体的量纲，不管是对服务组合，还是对服务评估都能够很好的量化，由此研究者提出了多种基于 QoS 的服务组合和评估优化模型，并用不同的方法去寻优得到好的结果；但缺点容易使服务组合或评估陷入局部最优，可能使系统负荷加重。建立评估因子方法的优点在于评估直观，评估因子涉及范围广，但评估因子建立困难，难于把握评估因子计算方法。而多指标体系的评估方法优点：不但从服务特征建立评估指标，还从服务状态建立了评估指标，同时建立了评估指标计算的多目标模型，但这种方法可能运算时间长。

表 3-30 中各类方法都表现出同一个问题，就是努力建立一种具有通用性的服务组合平台或框架。其中，基于 QoS 的框架方法是当前使用最多的方法，它的优点是能及时从服务质量的时间、花费、可靠性和可用性等角度动态跟踪服务组合效果，但在获得较优的结果存在难判断的弱点。基于形式化的框架方法可以较好的处理服务组合流程控制、识别等，但存在不易真正应用实现等缺点。中间件方法是一种常用的软件表现形式，是一种综合模式，任何一个相关的、好的服务组合方法都可以打包成一个中间件。

当前相关的服务研究，主要集中在服务发现和选择、服务组合及语义服务等研究方面。在本书中，从服务涉及的各个方面进行了分类和对比，得出当前采用最多的研究路线主要以服务质量（QoS）和语义服务为主。

3.4.6.2　主题图相关工作

主题图是一种类似于语义网络的知识表示模式，它提出了一种基于主题的元数据组织和描述方式，提供了语义级的数据导航和组织方式，是一个表达和交换结构化信息的元数据模型，是一种用于描述信息资源的知识结构的数据格式。它可以定位某一知识概念所在的资源位置，也可以表示知识概念间的相互联系，可以用于组织大量的信息，表达复杂规则和过程，管理分布知识和信息、门户功能等。它实际上是在信息资源的上层构建了一个结构化的语义网，独立于具体的技术平台[41]。其包括主题（Topic）、关联（Associations）、资源实体（Occurrence）、范围（Scope）、标记（Identity）、分面（Facet）等，其中主题、关联、资源实体和范围是核心要素，如图 3-80 所示。

主题图的应用目前已在信息组织、数字图书馆知识检索、门户集成系统和知识导航、企业信息系统集成、知识管理、e-Learning 和基于 Web 的信息传递系统中得到了应用[42]；且在这些领域体现出其应用灵活性、易操作性、针对性等优点。Ying Dongt 等人提供了一种面向主题图的 hyper-graph operations 知识管理方法[43]，其目的去引导下一代 Web 知识管理。在参

考文献[38]中把主题图应用去连接医学中的临床数据。Ralf Schweigert 等人提供了一种基于模式和系统规定参数的主题图融合和匹配方法，其采用多策略的匹配和配合的方法去发现基于语法的本体间的对应，或者主题图的语义特征和系统规定参数间的对应[44]。以及其他相关的信息组织和 Organizing learning materials[45][46]。

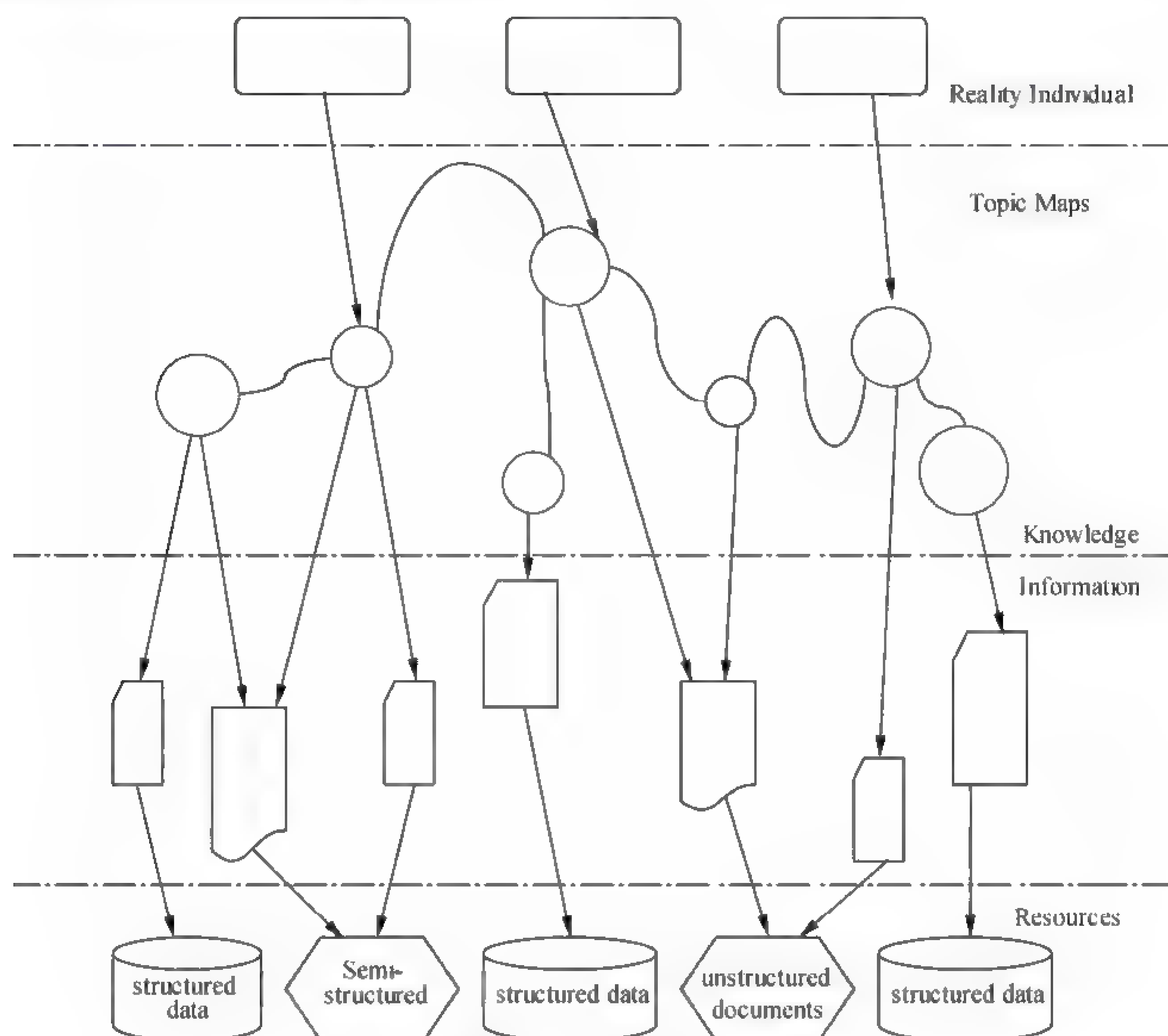

图 3-80　主题图结构图

在面向服务的语义化软件分析设计中以及语义 Web 服务+主题图的方法中，将结合现阶段 Web 服务相关研究的优点和主题图的现状，结合本体与语义 Web 服务描述间的语义关系，以及本体与主题图特征关系，并用这些关系建立起语义 Web 服务与主题图融合的切入点，从而实现 Web 服务与主题图的融合，即语义 Web 主题服务。通过主题图的作用有效实现 Web 服务精确导航定位其他有关联的 Web 服务，准确、快速完成服务组合[47]。因此需要从主题图和 Web 服务的语义去挖掘其融合关系，而 Web 服务是游离状态，具备具体的服务功能，主题图是这些服务功能状态知识语义网[41]。这时就出现了两个 Web 服务与主题图融合问题：

（1）Web 服务是处于主题图的上一层？

（2）Web 服务是处于主题图同一层？

在参考文献[41]中知，主题图是一个元数据模型、是一种描述信息资源的知识结构的数据格式；而在一个特定系统中，Web 服务组合就需要这些数据模型和数据格式，但 Web 服务描述与主题图采用的描述格式和方法是不同的，这时，就需要建立一种各描述方法的转换模

式来适应具有主题导航定位的服务组合。因此，我们可以认为在 Web 服务的“三角”结构中（三角指：服务消费、服务提供和 UDDL，以及服务功能描述的 WSDL），主题图数据描述是 WSDL 和 UDDI 的需求？但不能将一个 Web 服务等同于一个主题图。目前主题图通过 XTM 进行描述语义、语法关系，通过 TMDM（Topic Maps Data Model）和 TMRQL（Topic Map Relational Query Language）进行数据描述和查询，据此，要实现 Web 服务与主题图融合，必须解决至少两个方面的问题：

（1）解决 Web 服务与主题图的语义描述；

（2）解决识别映射。

因此 Nikita Ogievetskyt 等人创建了一个由 XTM 向 RDF 转换的在线转换器（XTM2RDF Translator）[48]①，通过采用 XLST 技术来实现 XTM 向 RDF 的映射，并且以 OWL-DL 构建了一种主题图的本体表示方法②。这些方法的出现，为实现主题图与 OWL 提供了理论和具体应用的支持。

而面向服务的语义化的软件分析设计研究的主要贡献有：

（1）概述当前 Web 服务及组合和主题图研究与就应用现状；

（2）分析主题图与 OWL、TMDM 和 WSDL 描述关系及融合方法，建立语义 Web 主题服务，并提出一种融合视图；

（3）提出一种基于语义蚁群算法的服务组合优化方法；

（4）建立一个本书方法的软件开发的开源平台；

（5）组织一组数据进行分析。

3.4.7 面向服务的软件语义的软件分析设计方法

前面所述详细构造了一种语义 Web 主题服务，它是以国内外现阶段面向服务计算（SOC）的研究现状为基础来进行描述的；同时，也以现阶段主题图的研究现状为基础，有效指出了现阶段主题图的所存在的问题。因此，以 SOC 和主题图的研究结果，提出了一种语义 Web 主题服务构造的思想，并以语义 Web+Web 服务为基础，引入了 Web+Web 服务+主题图的结构，把主题图的特征引入到了 Web 服务中，这样进一步丰富了语义 Web 服务的内容，提高了 Web 服务查找和定位的准确性。但这种构造方法是否合理，是否满足相关的语法和语义规则，针对这个问题，进行详细论述和说明，它们最终能通过 OWL 实现融合，同时，并根据国内外相关的文献资料，论述了该方法是可行的，是完全满足现阶段的语法和语义的要求。但怎样从逻辑上来实现融合的内部结构了，通过查阅文献，并进行相关研究得出，采用描述逻辑的 SHOIQ 方法能满足逻辑上的描述，这是因为 SHOIQ 与 OWL DL 是等价的。接着以 SHOIQ 为基础进行了语义 Web 主题服务内部结构描述，并结合了相关的描述结果。然后还实现了语义 Web 主题服务组合的模型，并根据 SHOIQ 的性质以及 Tableau 的规则，证明出了语义 Web 主题服务是满足 SHOIQ 和 Tableau 规则，最后并以实例进行了说明。

这些分析、研究结果直接就构造了一种面向服务的软件语义的软件分析设计方法，该方

① http://www.ontopia.net/topicmaps/materials/rdf.html

② http://www.mulberrytech.com/Extreme/Proceedings/html/2005/Cregan01/EMI2005Cregan01.html

法以语义 Web+主题图为基础，结合描述逻辑和本体加以实现。

小　结

本章全面概述和结合软件的分析设计方法，其内容包括了从开源软件应用角度概述了开源软件的分析设计的方法，也总结和分析了当前的面向对象和面向构件的软件分析设计方法，以及在这两种软件分析设计的基础上，概述了 UML，为后续全面分析面向的软件设计方法提供前期的基本知识。在本章的主要亮点之一就是分析研究了面向服务的软件分析设计方法中所涉及到的 SOA、BPEL、ESB 和 SoaML，并结合 UML 对这些面向服务的方法进行了详细的分析研究，同时也举例进行了详细进一步说明，这为理解这些方法提供了更为直观的方法。最后亮点提出了以语义 Web+主题图为基础，结合描述逻辑和本体加以实现的面向服务的软件语义的软件分析设计方法。

对这些软件分析设计方法的总结、分析研究，并进行体系化，为面向开源软件的软件分析设计方法提供了更直接、更可靠、更可行的指导和帮助。

参 考 文 献

[1] 石峰, 宋红. 面向对象方法. 北京：高等教育出版社, 2008.

[2] 黄柳青, 王满红. 构件中间：面向构件的方法与实践. 北京：清华大学出版社, 2006, 5.

[3] IBM. http: //www.ibm.com/developerworks/cn.

[4] Jack Park and Sam Hunting. XML Topic Maps: Create and using Topic Maps for Web. Addison-Wesley, 2002.

[5] Patil A, Oundhakar S, Sheth A, et al. Meteor-S Web Service Annotation Framework[C]//Proc. of the 13th International Conference on World Wide Web. New York, USA: ACM Presss, 2004: 17-22.

[6] Nikita Ogievetsky. XML Topic Maps through RDF Glasses. Markup Language: Theory & Practice, 2001, 3(3): 333-364.

[7] 史忠植, 常亮. 基于动态描述逻辑的语义 Web 服务推理. 计算机学报, 2008, (31)9: 1599-1611.

[8] Horrocks I，Patel. Schneider PF，Harmelen FV. From SHIQ and RDF to OWL：The making of a Web ontology language. et al of Web Semantics，2003，l(1): 7-26.

[9] ZHAO Tian-zhong, MIAO Zhuang, ZHANG YA-fei, et al. Reusing WordNet for Building Domain Ontology. Journal of System Simulation, 2007, 19(19): 4583-4586.

[10] Jung-Mn Kim, Hyopil Shin. Hyoung-Joo Kim. Schema and constraints-based matching and merging of Topic Maps. Information Processing and Management, 2007(43): 930-945.

[11] Ian Horrocks. Ulrike Sattler. A Tablean Decision Proceduce for SHOIQ. J Autom Reasoning, 2007(39): 249-276.

[12] Horrocks I, Sattler U, Tobies, S: Reasoning with individuals for the description logic SHIQ. In: McAllester, D. (ed.) Proceedings of the 17th International Conference on Automated

Deduction (CADE 2000). Lecture Notes in Computer Science, vol. 1831, pp. 482 496. Springer, Berlin Heidelberg New York (2000) .

[13] 邓水光，李莹，吴健，等. Web 服务行为兼容性的判定与计算. 软件学报, 2007, 18(12): 3001-3014.

[14] 王晓玲，郭志懋，周傲英. Web 服务组合的基于文法的消息处理. 计算机学报, 2005, 28(4): 478-485.

[15] Wen-Yau Liang, Chun-Che Huang. The generic genetic algorithm incorporates with rough set theory-An application of the web services composition. Expert Systems with Applications, 2009, 36(3): 5549-5556.

[16] Jun Yana, Ryszard Kowalczykb, Jian Lin, et al. Autonomous service level agreement negotiation for service composition provision. Future Generation Computer Systems, 2007(23): 748-759.

[17] John Debenham, Carles Sierra. Merging intelligent agency and the Semantic Web, Knowledge-Based Systems , 2008(21): 184-191.

[18] Zakaria Maamar, Soraya Kouadri Moste´ faoui, Hamdi Yahyaoui. Toward an Agent-Based and Context-Oriented Approach for Web Services Composition. IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, 2005, 17(5): 686-697.

[19] Brahim Medjahed, Athman Bouguettaya. A Multilevel Composability Model for Semantic Web Services. IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, 2005, (17)7: 954-968.

[20] KEITA FUJII and TATSUYA SUDA. SEMANTICS-BASED DYNAMIC WEB SERVICE COMPOSITION. International Journal of Cooperative Information Systems, 2006, 15(3): 293-324.

[21] Z. Maamar, N. C. Narendra*, S. Sattanathan. Towards an ontology-based approach for specifying and securing Web services. Information and Software Technology, 2006, (48)7: 441-455.

[22] Matteo Baldoni, Cristina Baroglio, Alberto Martelli, et al. Reasoning about interaction rotocols for customizing web service selection and composition. The Journal of Logic and Algebraic Programming, 2007(70): 53-73.

[23] Matteo Baldoni, Cristina Baroglio, Alberto Martelli, et al. Reasoning About Interaction rotocols for Web Service Composition. Electronic Notes in Theoretical Computer Science , 2004(105): 21-36.

[24] 范小芹，蒋昌俊，王俊丽，等. 随机 QoS 感知的可靠 Web 服务组合. 软件学报, 2009, 20 (3): 546-556.

[25] Jong Myoung, Chang Ouk, Ick-Hyun. Quality-of-service oriented web service composition algorithm and planning architecture. Journal of Systems and Software, 2008, 81(11): 2079-2090.

[26] Wen-Yau Liang, Chun-Che Huang , Horng-Fu Chuang. The design with object (DwO) approach to Web services composition. Computer Standards & Interfaces , 2007(28): 54–68.

[27] Patrick N. Blessa, Diego Klabjan, SooY. Chang. Heuristics for automated knowledge source integration and service composition. Computers & Operations Research , 2008(35): 1292-1314.

[28] Peter Hofner, Florian Lautenbacher. Algebraic Structure of Web Services. Electronic Notes in Theoretical Computer Science, 2008(200): 171-187.

[29] Jinghai Rao, Peep Kungas, Mihhail Matskin. Composition of Semantic Web services using Linear Logic theorem proving. Information Systems, 2006(31): 340-360.

[30] Zeng I，Benatallam B.. QoS-aware middleware for Web service com position. IEEE Transactions on Soft ware Engineering，2004, 30(5)：3ll-327.

[31] YANG Fang-chun, SHU Shen, LI Zhe. Hybrid QoS-aware model of semantic Web services composition strategy. Science in China Series E: Information Science, 2008, 38(10): 1697-1716.

[32] 杨文军，李涓子，王克宏. 领域自适应的 Web 服务评价模型. 计算机学报, 2005, 28(4): 514-523.

[33] 周相兵, 杨小平, 向昌成, 等. 面向本体的语义服务组合评价模型研究. 计算机集成制造系统, 2008, 14(12): 2346-2353.

[34] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, et al. A framework for QoS-aware binding and re-binding of composite web services. The Journal of Systems and Software, 2008(81): 1754-1769.

[35] Yu-Liang Chi, Hsun-Ming Lee. A formal modeling platform for composing web services. Expert Systems with Applications , 2008(34): 1500-1507.

[36] Shengwei Wang, Zhengyuan Xu, Jiannong Cao. A middleware for web service-enabled integration and interoperation of intelligent building systems. Automation in Construction , 2007(16): 112-121.

[37] Liangzhao Zeng, Boualem Benatallah, Anne H. H. Ngu, et al. QoS-Aware Middleware for Web Services Composition. IEEE TRANSACTIONS ON SOFTWARE ENGINEERING. 2004, 30(5): 311-327.

[38] K. Votis, C. Alexakos, B. Vassiliadis. An ontologically principled service-oriented architecture for managing distributed e-government nodes. Journal of Network and Computer Applications , 2008(31): 131-148.

[39] Jianqin Zhang, Jianhua Gong, Hui Lin, et al. Design and development of Distributed Virtual Geographic Environment system based on web services. Information Sciences , 2007(177): 3968-3980.

[40] Francisco Garcı´a-Sa´nchez, Rodrigo Martı´nez-Be´jar, Leonardo Contreras, et al. An ontology-based intelligent system for recruitment. Expert Systems with Applications , 2006(31): 248-263.

[41] Martin S. Laeher, Stefan Decker. RDF，Topic Maps，and the Semantic Web. Markup Languages. Theory&Pratice，2002, 3(3): 313-331.

[42] Steve Pepper. Article for the Encyclopedia of Library and Information Sciences. Bates &

Maack: ELIS, 2008. 10.

[43] Ying Dong, Mingshu Li. HyO-XTM: a set of hyper-graph operations on XML Topic Map toward knowledge management. Future Generation Computer Systems , 2004(20): 81-100.

[44] Ralf Schweiger, Simon Hoelzer, Dirk Rudolf, etc. Linking clinical data using XML topic maps. Artificial Intelligence in Medicine , 2003(28): 105-115.

[45] Jung-Mn Kim, Hyopil Shin, Hyoung-Joo Kim. Schema and constraints-based matching and merging of Topic Maps. Information Processing and Management, 2007(43): 930-945.

[46] B. -J. Shih, J. -L. Shih†, R. -L. Chen. Organizing learning materials through hierarchical opic maps: an illustration through Chinese herb medication. Journal of Computer Assisted Learning 2007(23): 477-490.

[47] 周相兵. 用描述逻辑实现语义主题 Web 服务组合的方法. 计算机应用, 2010, 30(10): 2763-2767.

[48] Nikita Ogievetsky. XML Topic Maps through RDF Glasses. Markup Language: Theory & Practice, 2001, 3(3): 333-364.

第 4 章　面向开源软件的软件开发方法

面向开源软件的软件开发方法与传统的软件开发是有区别的，但与传统的软件研发基本原理仍是存在一致性的，因此传统的软件开发同样可以指导面向开软件的软件开发。但又有别于面向构件、面向服务的软件开发方法。这是由于面向开源软件的软件方法主体是代码是开源的，所完成的功能是相对独立的，是实现某一领域需要的软件实体。因此，只要根据需求选择可靠性高、可维护性、成熟性好的开源软件作为软件的载体就可以有效提高软件开发效率、降低软件开发周期。

4.1　面向开源软件的软件开发特点

关于面向开源软件的软件开发特点在第 1 章进行了详细的描述。其特点主要体现在代码开源、代码免费、获取自由。在满足开源软件的发布协议许可下可以进行修改和改进。

4.1.1　软件体系架构选择原则

一个具体的需求，选择什么样的构架结构，从什么角度入手；怎样有效地对需求建模、怎样有效把握需求变化对整个软件构架的冲击，怎样有效控制整个项目在软件研发过程所面临的问题。这些问题往往是决定软件研发成功的主要因素，也是决定软件存在的周期性长短的关键。因此，可以参考以下原理进行按需求的软件架构：

（1）构架简单性。框架简单就是将软件开发过程各个环节实现简单化和有效化，并将这种观念渗入到软件开发过程中所有的工作中去。整个过程应该刚好生成足够完成工作的工件，开发人员应该尽量使用最简单的方法解决问题。而且应该构建一个清晰、简洁的解决方案。

（2）构架确定性。构架确定就是选择确定的软件框架模式来构架软件，来实现软件建模，不要因为需求变化而改变软件构架，也不要因为研发团队的人员变化也改变软件构架。除非的确该构架无法行通了，但此时，需要相关领域的权威专家再次认证后，确信无误后，方可以进行改变或调整现有的软件构架。

（3）构架可行性。不但要使所选择的软件构架具有简单性、确定性，还需要明确其可行性。而软件构架的可行性也是以预测为目的，以高效研发软件为前提，以获取恰当的软件生命周期为目的，以获得最佳的软件可靠性和可用性为最终结果。通常采用战略分析、调查研究、预测技术、系统分析、模型方法和智囊技术等得到软件构架的可行性。

4.1.2　面向开源软件的软件开发的代码原则

代码的优美、代码的简练、代码的高效执行、代码的可维护性、代码的可复用性等对面

向开源软件的软件整合开发具有重要的价值，也是影响开源软件的生命周期和可用性重要因数。下面从以下几个方面进一步说明代码在面向开源软件的软件开发中的代码原则。

（1）修补漏洞。许多开发方法可能不鼓励在过程中进行重构或变更，因为这些行为不直接用于产生客户代码。轻量级开发要求可以自由地修补太复杂或充斥 bug 的代码。需要为它做出预算。

（2）自动化单元测试。应该优先编写测试用例。可能还没有在测试第一的开发中成功过，但测试会间接带来重构代码的便利。以后会进一步讲述：广泛的单元测试改善客户体验，并提高代码的设计水平，这是因为它强迫解耦联系过于紧密的代码。

（3）使用短开发周期并积极调动客户参与其中。许多顶级的软件工作室通过剔除不必要的工件来简化开发周期。如果已经顺利得到客户的参与，那么很多的功能规范会变得越来越没必要。客户会很满意这种交互，并感激短周期开发，因为这稳步提高了客户的业务价值。

4.1.3 开源软件选择方法分析

目前，开源软件有十万级个，怎么从这些开源软件中选取可用性好、健壮性强的开源软件作为需求的选择是当前研发人员所面临的一个难点。近来，根据 ETH Life 报道，苏黎世（Zurich）工学院的科学家们最近的研究发现 Zipf 定律同样适用于开源软件，其 Zipf 定律最早在 1946 年出现在对英文单词出现的频率研究当中，它是一个经验定律。研究发现如果把单词出现的频率按由大到小的顺序排列，则每个单词出现的频率与它的名次的常数次幂存在简单的反比关系，这个定律后来在很多领域得到了同样的验证，包括网站的访问者数量、城镇的大小和每个国家公司的数量。如果以单词出现的频次将所有单词排序，用横坐标表示序号，纵坐标表示对应的频次，可以得到一条幂函数曲线。这个定律被发现适用于大量复杂系统[1]。

最近，苏黎世工学院的科学家对 Debian Linux 操作系统中软件包发行趋势进行研究，再一次发现了 Zipf 定律。科学家们对四个版本的 Debian Linux 的软件包进行了研究，其中最早的 Debian 版本只包括 474 个软件包，而最新的 Debian 版本则包括超过了 18000 个软件包。研究者通过每一个软件包的请求连接数研究其分布规律发现软件包和其请求连接数符合 Zipf 定律。同时，研究者还对每个软件包的请求数与时间的关系进行了研究，该分布和对应版本的操作系统的发行数量相关。

目前研究者们希望根据 Linux 软件包发布趋势的研究结果对企业风险进行了进一步研究。下面进一步叙述 Zipf 定律，使读者进一步明白该定律。

Zipf 定律是哈佛大学的语言学家 George Kingsley Zipf（IPA[zipf]）1949 年发表的①。它可以表达为在自然语言的语料库里，一个单词出现的频率与它在频率表里的排名成反比。所以，频率最高的单词出现的频率大约是出现频率第二位的单词的 2 倍，而出现频率第二位的单词则是出现频率第四位的单词的 2 倍。比如，在 Brown 语料库中，“the”是最常见的单词，它在这个语料库中出现了大约 7%（100 万单词中出现 69971 次）。正如齐夫定律中所描述的一样，出现次数为第二位的单词“of”占了整个语料库中的 3.5%（36411 次），之后的是“and”（28852 次）。仅仅 135 个字汇就占了 Brown 语料库的一半。

① http://www.nslij-genetics.org/wli/zipf

齐夫定律是一个实验定律，而非理论定律。齐夫分布可以在很多现象中被观察到。齐夫分布的在现实中的起因是一个争论的焦点。齐夫定律很容易用点阵图观察，坐标为 log（排名）和 log（频率）。比如，“the”用上述表述可以描述为 x = log(1), y = log(69971)的点。如果所有的点接近一条直线，那么它就遵循齐夫定律。最简单的齐夫定律的例子是“1/f function”。给出一组齐夫分布的频率，按照从最常见到非常见排列，第二常见的频率是最常见频率出现次数的 1/2，第三常见的频率是最常见的频率的 1/3，第 n 常见的频率是最常见频率出现次数的 $1/n$。然而，这并不精确，因为所有的项必须出现一个整数次数，一个单词不可能出现 2.5 次。然而，在一个广域范围内并且做出适当的近似，许多自然现象都符合齐夫定律。下面用数学公式表达 Zipf 定律如下①：

如果把单词出现的频率由高到低的次序排序，每个单词出现的频率与它的序号成简单的反比关系：

$$P(r) = \frac{C}{r^{\alpha}} \tag{1}$$

式中：$C \approx 0.1$，$\alpha \approx 1$。这就是简单的 Zipf 定律，如果对词频及其序号作对数－对数（log-log）曲线，将会出现一条直线，并且斜率≈1。Zipf 定律表明在英语单词中只有少数的词被经常使用，而绝大多数词则很少被使用。同样，开源软件也只有部分被经常使用。很大一部分开源软件很少被大范围的使用。如本书的 SSH 就是被经常使用的。

类似地 Pareto 研究了个收入的统计分布，发现少数人的收入要远高于大多数人的收入。若把个人收入 X 高于某人特定的值 x 的概率表示为

$$P[X \geqslant x] = \left(\frac{m}{x}\right)^{k} \sim \frac{1}{x^{k}} \tag{2}$$

式中：$m>0$，$k>0$，$x \geqslant m$。这样收入低于某个特定值 x 的概率为 $P[X < x] = 1 - \left(\frac{m}{x}\right)^{k}$，$m$ 表示最低收入。

假设收入正好是 x 的概率密度是 $p(x)$。则

$$P[X < x] = \int_{m}^{x} p(x)dx = 1 - \left(\frac{m}{x}\right)^{k}$$

并且可以解得

$$p(x) = km^{k}x^{-(k+1)} \sim x^{-(k+1)} \sim \frac{1}{x^{\beta}} \tag{3}$$

式中：$\beta = k+1$，称为幂律分布。

假设在英语中存在 Zipf 定律，第 r 位单词出现的频率为 $f = C_1 r^{-\alpha}$。则存在 r 个单词词频高于 f，对应概率为 $P[X \geqslant f] = C_2 r$。即 $P[X \geqslant f] = C_2 (C_1)^{\frac{1}{\alpha}} (f)^{-\frac{1}{\alpha}}$；式中，$C_1$、$C_2$ 是归一化常数，这时词频正好为 f 的概率为

$$p(f) = f^{-(\frac{1}{\alpha}+1)} \sim f^{-\beta} \tag{4}$$

① http://www.hpl.hp.com/research/idl/papers/ranking/ranking.html

式中：$\beta = \frac{1}{\alpha}+1$，由于$\alpha$=1，所以$\beta$=2。

下面是以随机文本模型来进一步说明 Zipf 定律，它是在 Zipf 中是一种较有影响的模型。假设语言是 M+1 种符号产生的连续符号序列，其中 M=26 表示每个单词是由 a～z 的 26 个英文字母组成，第 M+1 种符号是空格，在连续符号序列中区分不同的单词。显然每个单词的长度可以是由 1 个字母到无穷个字母组成。在随机文本中任何一个字母后面跟随 26 个字母中任意一个字母都是等概率的；而人类语言则不同，不同字母组合相差概率很大，如字母 q 后面只能和字母 u 组合，元音字母 a、e、i、o 后面可以与几乎所有 26 个字母组合。但数值实验表明，随机文本模型与自然语言一样都在 Zipf 定律。

这时，假设空格与任意字母出现的概率都是相同的，即：p=1/27，这样出现字符串_a_的概率为：$(1/27)^3$。出现字符串_bbs_的概率为$(1/27)^5$。某个长度为 L 单词的概率为

$$P_i(L)=\frac{C}{(M+1)^{l+2}},\quad i=1,2,\ldots,M^L \tag{5}$$

式中，M^L 表示有 M^L 个不同单词长度为 L，以为归一化常数，即采用所有不同长度单词概率和为 1。即如下式：

$$\sum_{L=1}^{\infty} M^L \frac{C}{(M+1)^{M+2}}=1 \tag{6}$$

且利用级数求和可得关系

$$\sum_{L=1}^{\infty} M^L \frac{C}{(M+1)^{M+2}}=\frac{M}{(M+1)^2}=1$$

则求得归一化因子为

$$C=\frac{(M+1)^2}{M} \tag{7}$$

这时，出现某长度为 L 单词的概率为

$$P_i(L)=\frac{1}{M(M+1)^L} \tag{8}$$

则就可以得到发现长度为 L 单词的概率为

$$P(L)=M^L P_i(L)=\frac{M^{L-1}}{(M+1)^L} \tag{9}$$

根据以式（9），相同长度 L 的单词具有相同的概率，并且单词越长，其出现的概率就越低。这样随机文本中单词长度为 L 的排序决定了单词的词频排序 $r(L)$，并且存在如下关系：

$$\sum_{l=1}^{L-1} M^l < r(L) < \sum_{l=1}^{L} M^l \tag{10}$$

如：0<r(1)≤1，M<r(2)≤M+M^2 等。这时，利用求各公式可得 $\sum_{l=1}^{L} M^l = \frac{M(M^L-1)}{M-1}$，则这时公式变换有

$$\frac{M(M^{L-1}-1)}{M-1} < r(L) < \frac{M(M^L-1)}{M-1} \tag{11}$$

将式（11）变换可得

$$L-1<\log_M\left(\frac{M-1}{M}r(L)+1\right)\leqslant L \tag{12}$$

所以

$$\frac{1}{(M+1)^{L-1}}>\left(\frac{1}{M+1}\right)^{\log_M\left(\frac{M-1}{M}r(L)+1\right)}\geqslant\frac{1}{(M+1)^L} \tag{13}$$

将式（13）各项乘以 1/*M*，可得

$$\frac{1}{M(M+1)^{L-1}}>\frac{1}{M}\left(\frac{1}{M+1}\right)^{\frac{\lg(M+1)}{\lg M}}\geqslant\frac{1}{M(M+1)^L} \tag{14}$$

可进一步将式（14）改写如下：

$$P_i(L)\leqslant\frac{C}{(r(L)+B)^{\alpha}}<P_i(L-1) \tag{15}$$

式中：$\alpha=\frac{\lg(M+1)}{\lg M}$，$B=\frac{M}{M-1}$，$C=\frac{M^{\alpha-1}}{(M-1)^{\alpha}}$。这时就可以得到广义的 Zipf 定律：

$$P(r)=\frac{C}{(r+B)^{\alpha}} \tag{16}$$

通过诸如 Excel、Origin、Matchematica 等数学软件就可以获得一组参数：如，当 M=26 个字母时，就可以得到α=1.01158，B=1.04，C=0.04。

上面完成 Zipf 经验定律的两个非常有用的结论，通过这个理论进行举一反三就可以从数学的角度获得开源软件的 $P(r)$。因此，通过 $P(r)$和需求就可以选择到一款较为满意的开源软件或开源软件包。

4.2　面向开源软件的软件开发方法

由于开源软件多为采用传统的软件开发方法进行，这就决定传统的软件开发方法同样适用于面向开源软件的软件开发。这是因为，面向开源软件的软件开发也可以认为是一种软件开发方法。在软件效率方法，开源软件的软件方法可以提升软件开发效率，减少代码重复开发，聚集多人编程智慧，简化软件研发过程。

4.2.1　开发模型分析

软件开发模型（Software Development Model）是指软件开发全部过程、活动和任务的结构框架。软件开发包括需求、设计、编码和测试等阶段，有时也包括维护阶段。软件开发模型能清晰、直观地表达软件开发全过程，明确规定了要完成的主要活动和任务，用来作为软件项目工作的基础。传统的软件开发模型通常有瀑布模型（waterfall model）、渐增模型/演化/迭代（incremental model）、原型模型（prototype model）、螺旋模型（spiral model）、喷泉模型（fountain model）、智能模型（intelligent model）、混合模型（hybrid model）[2]。

而面向开源软件的软件开发同样可以采用这些传统模型来指导，分析表明，其中的演化

模型更适合面向开源软件的软件开发。这时因为可以根据不同的需求，可以选择不同的开源软件来实现按需的软件开发。当然面向开源软件的软件开发模型正在处于发展阶段，即采用什么样的模型来指导其软件开发，目前尚未形式一个统一的模型。但有相关组织正在努力来完成这一目标，它们首先从开源软件的可信性和质量入手来逐步加以解决，相关开软件的可信和质量在后面有叙述。

4.2.2 开发需求分析

需求分析指的是在创建一个新的或改变一个现存的系统或产品时，确定新系统的目的、范围、定义和功能时所要做的所有工作。需求分析是软件工程中的一个关键过程。在这个过程中，系统分析员和软件工程师确定顾客的需要。只有在确定了这些需要后他们才能够分析和寻求新系统的解决方法[2]。

长期以来，顺利地完成需求分析是一个艰巨的挑战。首先要确认所有持有关键信息的人本身就不容易，然后还要从这些人获得可用的信息，把这些信息转化为清晰的和完整的形式。同时分析者还要考虑到可能的限制。除此之外他们还要考虑一个项目的是否可行、是否在规定的时间里可以完成、价格上是否负担得起、是否合法、是否符合道德。这就直接导致了需求分析有可能在一个项目中成为一个漫长、艰巨的工作。需求分析专家与他们的顾客交谈、记录他们的交谈结果、分析他们收集的信息，从中提取互相矛盾的地方，总结出一个总体观念，然后再与顾客交谈他们发现的问题。这个过程可以不断重复，在有些项目中这个过程可以伴随着整个生命周期。

而面向开源软件的软件开发，同样需要逐渐明确需求，进行全面的需求分析。作为软件研发本身来讲，传统的需求分析方法同样可以适用面向开源软件的软件开发，这是因为，仅仅只将开源软件作为一种开发工具或平台，或者一种方法，像 Java 语言一样使用。但将开源软件作为一种系统方法或者一种新生的软件开发原理时，这种情况就发生了变化，因为就可能成为一种实实在在的新的软件工程学或系统学。这时就需要从这种新的角度来重新审视面向开源软件的需求分析方法。

4.2.3 开发分析设计方法

软件分析设计就是在满足软件体系结构和需求的情况下，是对软件的具体实施。其分析就是根据软件构架和需求进行软件具体的分析，而设计在建立分析的基础之上，即通过软件分析结果，再进行具体的功能设计等。软件的分析设计是实现软件的具体操作，是完成软件的具体实施过程。而对开源软件的软件开发来讲，若从软件开发本身角度来讲，同样具有这样的特性，若把面向开源软件的软件开发作为一种软件工程学或软件体系学来讲，就不但具有这样的特性，还应该具有工程和体系的特征。即面向开源软件的软件开发方法可以具有传统的软件分析设计方法，也可以把开源软件当成一种软件工程学。

但把面向开源软件的软件开发方法时，不能像传统方法那样进行分析设计，而是在进行分析设计，必须兼顾考虑开源软件的结构和方法，以免使得各种结构、方法与技术兼容。这一点面向开源软件的软件开发主要注意的问题之一。同时，当把面向开源软件的软件开发当

成一种工程学来指导软件开发时，需要把开源软件提升至一个全局的角度来分析，即通过将开源软件及开源软件的包进行构架重组，按需生成新的软件系统。

4.2.4　开发实现流程

软件开发流程（Software development process）即软件设计思路和方法的一般过程，包括设计软件的功能和实现的算法和方法、软件的总体结构设计和模块设计、编程和调试、程序联调和测试以及编写、提交程序。而软件项目生命周期刻画了一个工程从起始到完成，是如何进行计划、控制和监控的模型。传统方法通常采用需求调研分析、概要设计、详细设计、编码、测试、软件交付准备、验收、维护等流程来实现软件。

而面向开源软件的软件的开发流程，也可以采用传统的方法来实施，但在实现这个流程时，需要考虑和考察开源软件的特性，即把开源软件当成什么的模式来实现软件研发，这些模式主要包括将开源软件作一种软件开发工具、作为一种软件开发方法、作为一种软件开发技术、作为一种软件开发语言。因此，需要根据这些模式来确定其软件开发流程。

4.2.5　测试方法

软件测试（Software Test）的经典定义是在规定的条件下对程序进行操作，以发现程序错误，衡量软件品质，并对其是否能满足设计要求进行评估的过程。但通常软件测试描述一种用来促进鉴定软件的正确性、完整性、安全性、和品质的过程。由于软件测试永远不可能完整的确立任意电脑软件的正确性，因此可以换句话说，软件测试是一种实际输出与预期输出间的审核或者比较过程。

软件测试一度被认为是编程能力偏低的员工的工作，直到今天，仍然有许多公司把优秀的人才放在编码上，也有更多公司让优秀的人才进行设计，可是很少公司让优秀的人才进行测试工作。实际的软件工程实践证明，让对软件思想有深刻理解的工程师进行软件测试，可以大幅度的提高软件质量。表 4-1 所示是按测试类型可以分为如下。

表 4-1　主要软件测试方法分类

类型	项	简单描述
测试进程	Alpha 测试	通常是在软件由潜在用户/客户或一个独立的测试团队，且在是阶段性的开发完成后开始进行
	Beta 测试	当 Alpha 阶段完成后，开发过程进入到 Beta 阶段。一般认为 Beta 测试就是由一部分受控制的客户进行的黑盒测试
	Gamma 测试	该测试阶段对应的是对“存在缺陷”产品的测试
测试方法	黑盒测试	这种测试不需要了解软件的内部构造，是从用户的角度对程序进行的测试，只知道程序的输入（将测试数据输入到软件）、输出（确认输出结果是否正确）和系统的功能就可以
	白盒测试	白盒测试又称结构测试和逻辑驱动测试。白盒测试法的覆盖标准有逻辑覆盖、循环覆盖和基本路径测试
测试类型	功能测试	按照测试软件的各个功能划分进行有条理的测试，在功能测试部分要保证测试项覆盖所有功能和各种功能条件组合

续表

类型	项	简单描述
测试类型	系统测试	对一个完整的软件以用户的角度来进行测试
	极限值测试	对软件在各种特殊条件，特殊环境下能否正常运行和软件的性能进行测试
	性能测试	性能测试是对软件性能的评价
测试阶段	单元测试	单元测试是最微小规模的测试;以测试某个功能或代码块。典型地由程序员而非测试员来做，因为它需要知道内部程序设计和编码的细节知识
	集成测试	是指一个应用系统的各个部件的联合测试，以决定他们能否在一起共同工作并没有冲突
	系统测试	系统测试主要包括功能测试、界面测试、可靠性测试、易用性测试、性能测试
	回归测试	回归测试指在软件维护阶段，为了检测代码修改而引入的错误所进行的测试活动

当把面向开源软件的软件开发当成一种方法，这些主要的测试方法同样适合于面向开源软件的软件开发。

4.3 面向开源软件的软件开发标准探索

当把面向开源软件的软件开发当成一种软件工程学、一种新的软件开发时。然而目前面向开源软件的软件开发是正在发展的一个方法，很多原理、方法和技术仍处于发展阶段，尚未形成一个面向开源软件的软件开发标准，因此，本节根据软件工程学与软件开发方法为基础，试着探索一种面向开源软件的软件开发标准。

4.3.1 软件可信性

前面的章节已经较全面的叙述、分析和总结了软件的可信性，而开源软件的可信性更为重要，这是因为很多开源软件是由个人或团队研发而成（但也不能说没有可信性），这就使得开源软件可信性更自由、更难把握确切的可信性。所以研究开源软件的可信性具有重要的意义，使得只要确定了开源软件的可信性，就可以有效使用开源软件实现软件构架和研发，从而体现开源软件的价值。图 4-1 所示的可信性原模型可以用于指导开源软件的可信性。

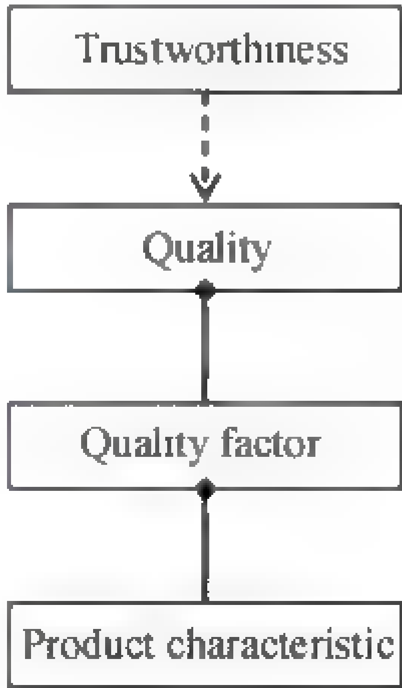

图 4-1　可以用于指导开源软件的可信性元模型

通常对软件可信性进行评估和考量时，需要所依赖的事实，称为可信证据：这些证据可以包括开发阶段证据、提交阶段证据和应用阶段证据，这样就可以从三方面来收集可信性证据。而作为互联网上的开源软件多数没有开发阶段和提交阶段的证据，而这些没有开发阶段和提交阶段证据的软件也可能在某方面具有较好的可信性，因此

按上述方式搜集证据不利于开源软件的可信评估，也不利于软件制造者个性特长的发挥。同时，开源软件常被用户和开源爱好者分析和改进从而提高软件质量和可信性，而这些改进在上述可信证据中难以体现，因此其不利于软件的可信演化；并且人在认识软件的时候，通常考虑较多的是其功能和体系结构，而较少想到软件生命周期各个阶段的具体过程是怎样的，因此上述的证据分类方式与人对软件的认知习惯不相符，不利于基于认识与理解的信任的形成[3]；但可以从图 4-2、图 4-3 所示来评估开源软件的可信性。

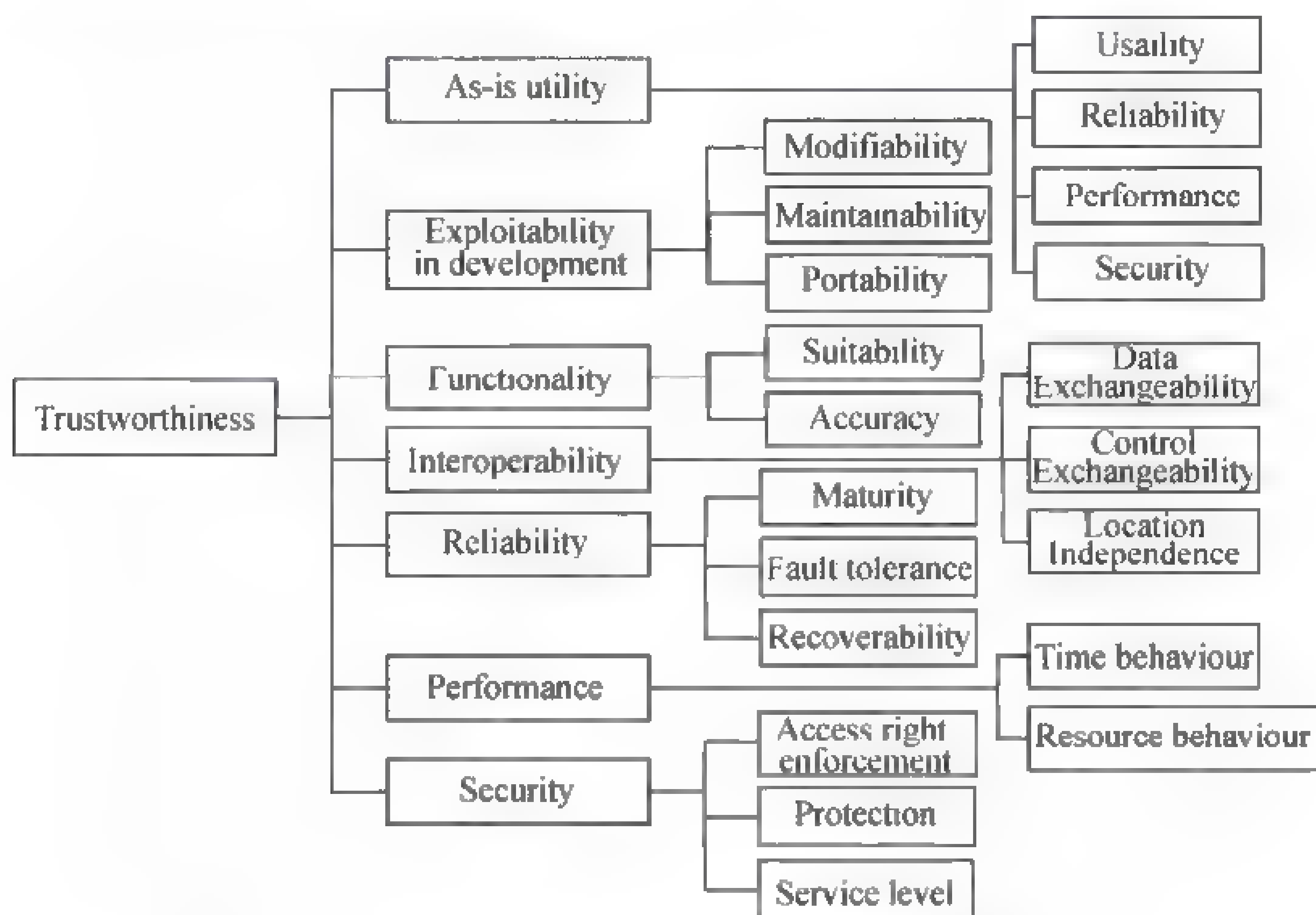

图 4-2　可以用来度量开源软件的可信性概念模型（1）

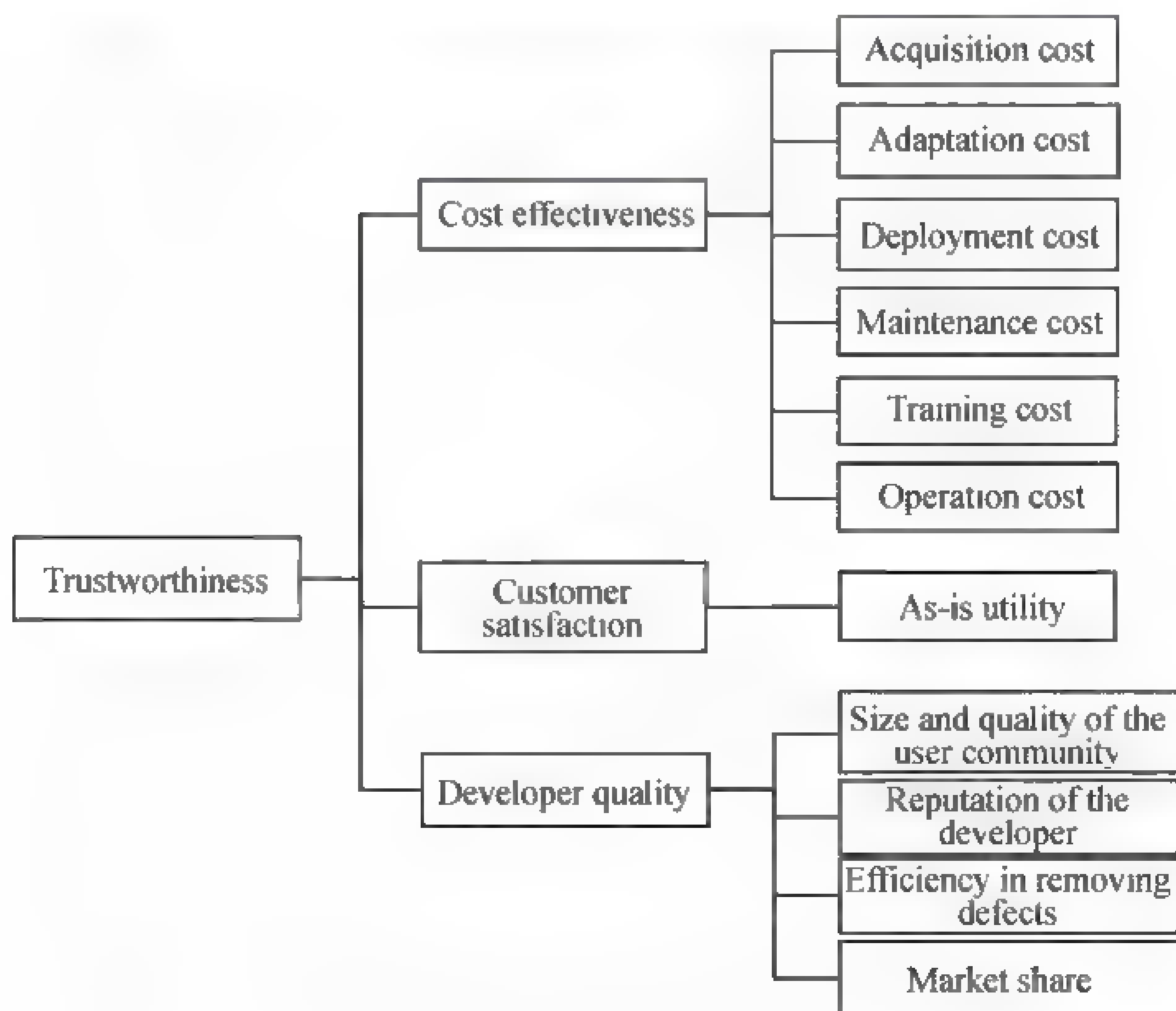

图 4-3　可以用来度量开源软件的可信性概念模型（2）

在图 4-2 中从开源软件的本身所需要满足的属性列举了可信性的评估因子，图 4-3 中从开源软件的非功能性和客户的角度列举了可信性的评估因子。根据图 4-2 和图 4-3 可以进一步得到图 4-4 所示的开源软件可信性评估结构。

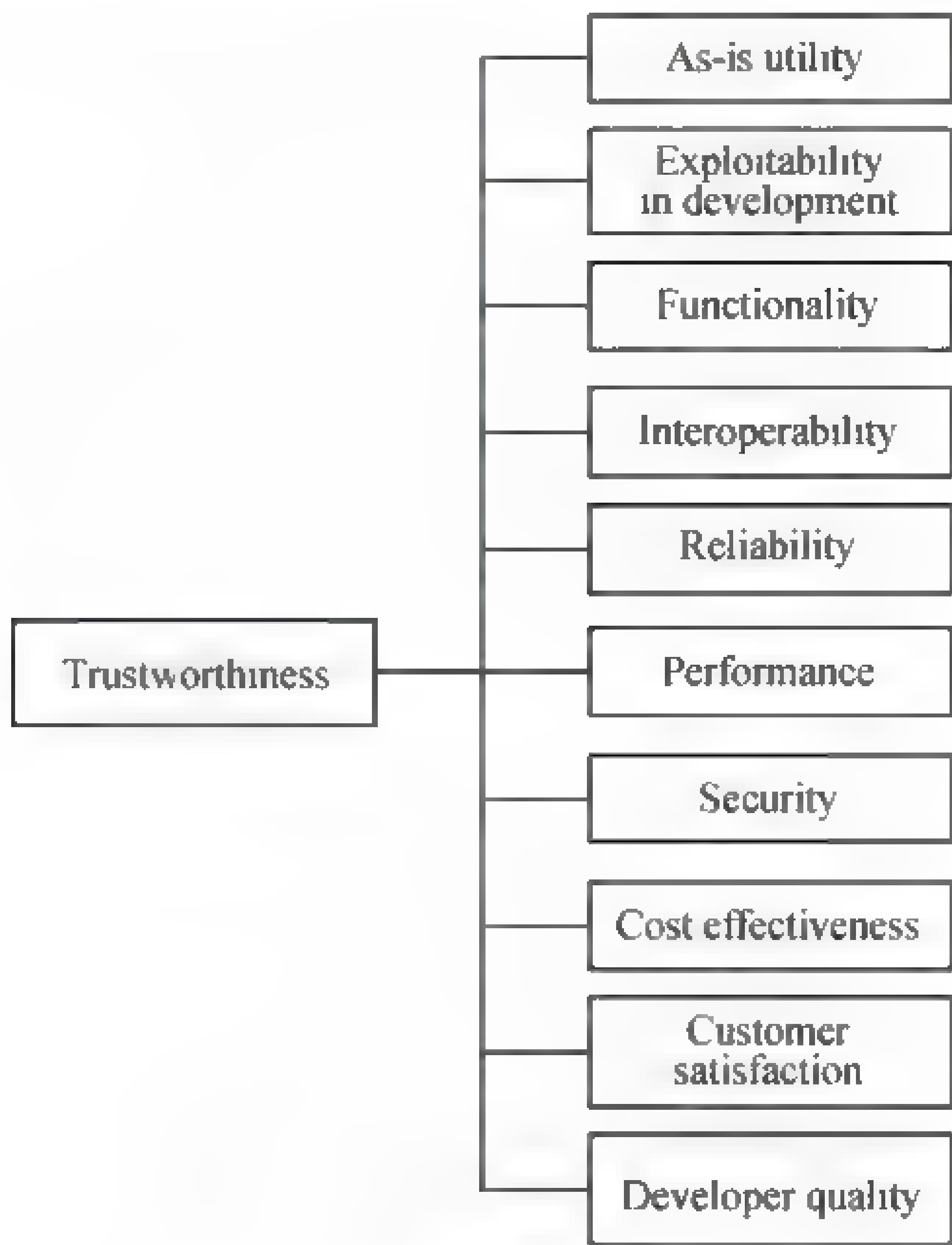

图 4-4 开源软件的可信性结构

4.3.2 软件质量

开源软件和传统商业软件在软件成熟度评估的方法上有重要的不同。传统软件由于无法了解软件内部的情况，软件的评测主要依赖与黑箱测试的方法。开源软件在这方面有明显的好处。评测人员可以深入分析软件的架构和技术特点，亲自或使用代码评测软件来对程序进行分析，能够发现程序中隐藏的代码错误，分析代码的复杂度、耦合性等程序特性，给开发者提供非常有价值的技术参考，帮助开发者有效提高软件的开发质量。这一点是开源软件质量评估中最大的优势之一。开放源代码软件质量特征按照软件质量国家标准 GB/T8566—2001G，软件质量可以用下列特征来评价：功能特征、可靠特征、易用性特征、效率特征、可维护特征、可移植特征。其中每一个质量特征都分别与若干属性相对应。

在开放源代码软件的质量评估中，除了要考虑以上一般性软件质量特性外，还应看到，由于开放源码软件开发模式和商业模式的独特性，一些重要的软件自身或非自身因素也必须加入到评估的标准中来。一般这些因素包括技术架构、代码质量、开发模式、社区建设、商业支持和法律问题等。

开放源代码软件由于其开发模式和运作模式的独特性，软件带有鲜明的特点。各开源软件由于其开发目标不同，技术实力各异，开发管理的差别以及应用推广力度的大小，使得软件质量与应用成熟度方面千差万别，不仅给用户选择带来了很多不便，也降低了企业在其商

业环境中使用开源软件的意愿。

开放源代码软件由于开发模式和运作模式的独特性，其软件带有鲜明的特点。开放源码软件成熟度评估的方法需要实践中不断的探索[1]。可以从以下几个方面来探索：

（1）开放源代码软件与传统非开放源码软件的重要不同点在于其技术方案和源代码的开放性。因此，开放源码软件的质量评估的一个重要目标是分析软件的技术架构和技术路线，得出软件在架构的合理性、技术路线的可行性、技术的先进性等方面的评估看法，使得评估能够从宏观上，从技术上对软件的成熟度给出相应的评价。

（2）开放源码软件对于评估人来说最大的好处是，可以直接阅读分析软件的代码，通过这一最直接的方式，可以得出准确的、公正的判定。因此，提供对代码质量和代码可信度的检测是开放源码评估中的重要环节。

（3）软件易用性和可用性的评估是任何软件成熟度评估中重要的指标，是实现开源软件成熟度评估所要实现的重要目标之一。该目标的实现，对于评估的实际参考价值的多少非常重要，是用户及其系统集成商重点关注的内容。

（4）软件应用的成熟度以及软件商业或非商业应用的支持力度能够从侧面反映软件的发展状态。在开源软件中，应用越广泛，社区支持或商业支持越好的软件能够吸引更多的参与者，反过来能够更好的促进软件的发展。所以，开源软件成熟度评估所要达到的一个重要目标是提供软件应用与支持的公正评价。

（5）由于开放源代码软件可能采用多种版权协议，某些软件有可能或有潜在的可能性涉及其他版权协议或技术专利等问题需要我们加以评估，是否会对软件的发展形成正面的或负面的影响。

4.3.3　软件复用

软件复用是一种计算机软件工程方法和理论。20 世纪 60 年代的软件危机使程序设计人员明白难于维护的软件成本是极其高昂的，当软件的规模不断扩大时，这种软件的综合成本可以说是没有人能负担的，并且即使投入了高昂的资金也难以得到可靠的产品，而软件重用的思想是解决这一问题的根本方法。软件复用的主要思想是将软件看成是由不同功能部分组件所组成的有机体，每一个组件在设计编写时可以被设计成完成同类工作的通用工具，这样，如果完成各种工作的组件被建立起来以后，编写一特定软件的工作就变成了将各种不同组件组织连接体来的简单问题，这对于软件产品的最终质量和维护工作都有本质性的改变。

而开源软件从软件复用角度来讲，可以有效实现软件的复用，即一个可信、质量好的开源软件可以重复的应用。所以开源软件的发展和步骤的成熟，能积极推动软件复用的水平提高。分析认为，开源软件的软件复用较构件技术先进，这是因为开源软件是开源的特征，构件技术是无法比拟的。

4.3.4　软件再生

软件再生理论认为，计算系统运行过程中的系统资源损耗是影响系统性能的主要因素。

---

① http://www.csip.org.cn/download

并且软件构造越来越复杂，由于软件长时间运行过程中伴随着使用完的资源得不到释放，内存的泄漏和溢出等现象，导致系统性能下降，甚至崩溃。针对此，相关研究认为可以通过有计划地释放系统资源在一定程度上恢复系统性能。

而开源软件的软件再生，就是针对开源软件状态，通过群体智慧不断更新软件，使开源软件越来越成熟可靠、可信和具有高质量，并在这种更新中有越来越多的使用者。并且在相当长的一段时间内有使用者提出不同的改进意见，而又在后一段时间内使用者提出的意见越来越少。这就表明开源软件再生能力越来越强，生命周期越能满足不同的需求变化，如图 4-5 所示。

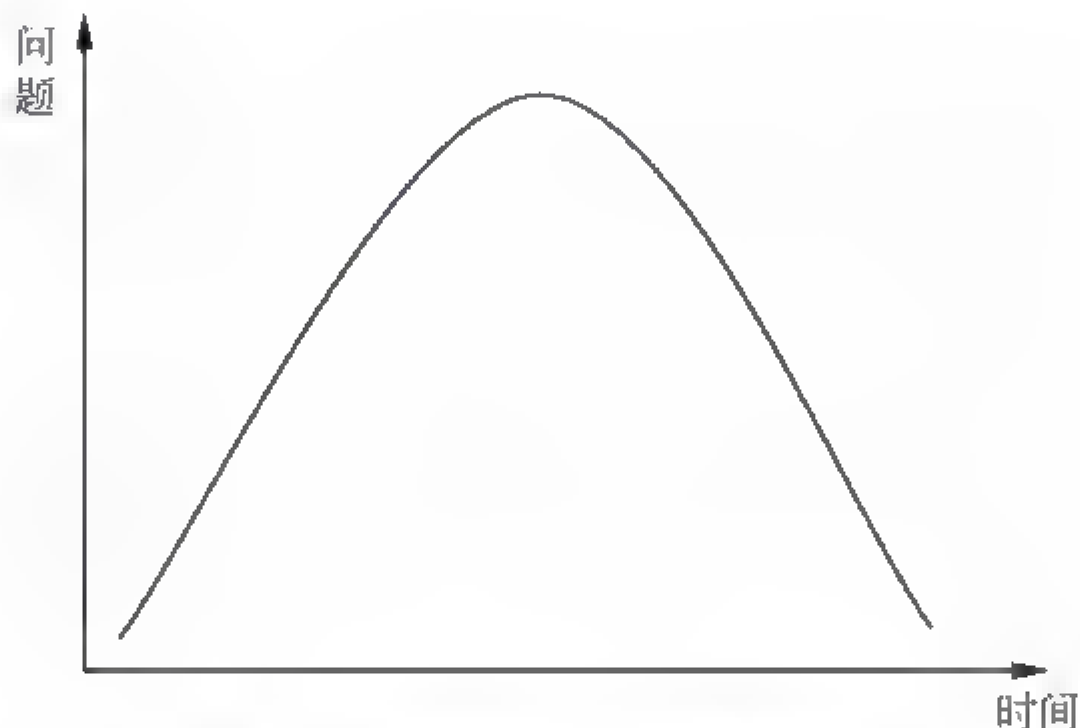

图 4-5　开源软件的再生影响曲线

同时，从另一方面，开源软件的再生还体现在软件自身合理性、完整和可用性上，这样的开源软件更易于被开源社区接纳，也更易于在开源社区中引起重视。从而让更多的人参与到这项工作来。也就是说没有再生能力的开源软件是不可能在开源社区中生存，也不能吸引有效的用户来开发和使用这样的开源软件。一款开源软件要有创新性、可用性、开放性和实效性，才能使该开源软件具有再生能力。

4.3.5　软件自动化

软件自动化一般认为包括程序设计自动化和文档设计自动化两部分，也就是说，软件自动化的目标是根据用户的需求来实现软件自动生成，即只要软件的需求原型确定就可以自动生成一款软件原型系统。只要在原型系统的基础上实现需求变化，相应的软件系统也变化，这种模式与当热门的 SAAS 很相似。

而开源软件的自动化就是在满足可信性、质量和再生的原则下，实现开源软件按需求在 Internet 下自动实现搜索和聚焦，从而找到可靠性高和可用性好的开源软件系统作为软件各方人员使用。

4.3.6　软件验证与确认

验证和确认产生于 20 世纪 70 年代，是美国航天局高可靠性软件系统生产实践的一个产物。软件设计在得到实现后，是否完全达到设计的目标、是否满足客户的真正需求需要通过

验证和确认才能给出准确的结论。软件系统的验证、确认贯穿整个软件生命周期，并占有重要地位。其：验证是指检验软件是否已正确地实现了产品规格说明书所定义的系统功能和特性，验证过程提供证据表明软件相关产品是否与生命周期活动的要求相一致、是否满足生命周期过程中的标准、实践和约定。验证为判断每个生命周期活动是否已经完成以及是否可以启动其他生命周期活动建立一个新的基准。确认是为了保证所生产的软件可追溯到用户需求的一系列活动，确认过程提供依据表明软件是否满足客户需求，并解决了相应的问题[4]。

而做开源软件的验证与确认不但要满足上述要求，还需要检查是否满足开源软件的许可协议。也是开源软件的可信性和再生性的具体的体现，特别是对开源软件的再生性具有重要的作用和意义。

小　结

本章阐述和分析了面向开源软件的软件开发，其结合传统的、经典的软件工程思想来分析面向开源软件的软件开发。特别地采用了 Zipf 定律来实现分析开源软件的选择，同时也简要分析了开源软件可信性和质量，并以此提出了开源软件的再生性概念等。但将开源软件作为一种新型的软件开发，还有需要更多原理和方法来支撑，毕竟一种新的软件开发方法需要经过一定过程才能从提出走向成熟，而且这个路非学艰难，也有可能走不通。可以预测：面向开源软件的软件开发总是有机会的，但其基本前提是必须对不定期对开源软件进行分析分类，进行评估甄别，最终这些有效的开源软件及开源软件包以 SaaS 发布出去，实现开源软件服务化，可访问化。

参 考 文 献

[1] Lü L, Zhang Z-K. Zhou T (2010) Zipf's Law Leads to Heaps' Law: Analyzing Their Relation in Finite-Size Systems. PLoS ONE 5(12)：e14139. doi:10.1371/journal. pone. 0014139.

[2] 张海藩. 软件工程导论（第 5 版）. 北京：清华大学出版社，2008.

[3] DING Xuelei, WANG Huaimin, WANG Yuanyuan,et al. Verification oriented trustworthiness evidence and trustworthiness evaluation of software. Journal of Frontiers of Computer Science and Technology，2010，4(1)：46-53.

[4] 高月，梁成才，王川，等.软件验证与确认在 MIS 开发中的应用.计算机工程，2011，37(1)：84-86.

第 5 章　面向开源软件的软件开发技术

前面四个章节，全面总结、概述、分析研究了面向开源软件的软件开发方法所需要的体系结构、分析设计方法，以及实现方法与策略。下面主要以这些技术原理和框架结构为基础进行总结、概述和分析面向开源软件的软件开发技术。

5.1　概述

面向开源软件的软件开发技术就是用来实现开源软件的技术，而且有些技术本身也叫开源软件。目前主要 Java、PHP、Perl、C、C++等作为实现开源软件的主要技术，其中 Java 是目前开源软件实现的最普通的技术。这些技术（包括开源软件本身）就形成了当前几十万种以上的开源软件阵营，而且这种势态还正在发展。这样就使得开源软件产业的发展是势不可挡的，技术进步快、技术共享等特性，以及越来越多的产业资源的支持和快速发展的市场，都是开源软件得以持续发展的有利条件。

5.2　常用的开发及平台语言

目前很多开源软件实现和应用语言都摆脱了硬件束缚，使其直接运行在硬件之上，让软件开发的效率获得了提升，也简化也了程序员的工作流程。下面就选择几种典型的开源软件的语言及平台来进行概述。

5.2.1　PHP

PHP 是英文超级文本预处理语言 Hypertext Preprocessor 的缩写，它是一种 HTML 内嵌式的语言，是一种在服务器端执行的嵌入 HTML 文档的脚本语言，语言的风格有类似于 C 语言，被广泛的运用。PHP 也是一种在计算机上运行的脚本语言，主要用途是在于处理动态网页，也包含了命令行运行接口（Command Line Interface），或者产生图形用户界面（GUI）程序。

PHP 最早由 Rasmus Lerdorf 在 1995 年发明，而现在 PHP 的标准由 PHP Group 和开放源代码社区维护。PHP 以 PHP License 作为许可协议，不过因为这个协议限制了 PHP 名称的使用，所以和开放源代码许可协议 GPL 不兼容。但 PHP 的应用范围相当广泛，尤其是在网页程序的开发上。一般来说 PHP 大多运行在网页服务器上，通过运行 PHP 代码来产生用户浏览的网页。PHP 可以在多数的服务器和操作系统上运行，而且使用 PHP 完全是免费的。根据 2007 年 4 月的统计数据，PHP 已经被安装在超过 2000 万个网站和 100 万台服务器上。

PHP 独特的语法混合了 C、Java、Perl 以及 PHP 自创新的语法。PHP 可以比 CGI 或者 Perl 更快速的执行动态网页。执行效率比完全生成 HTML 标记的 CGI 要高许多；PHP 还可以执行编译后代码，编译可以达到加密和优化代码运行，使代码运行更快。PHP 具有非常强大的功能，所有的 CGI 的功能 PHP 都能实现，而且支持几乎所有流行的数据库以及操作系统，包括数据库有 MySQL、PostgreSQL、Oracle、Sybase、Informix 和 Microsoft SQL Server 等，包括的操作系统有 Unix、Windows 等。

5.2.1.1　PHP 发展进展

PHP①是 Rasmus Lerdorf 为了要维护个人网页，而用 C 语言开发的一些 CGI 工具程序集，来取代原先使用的 Perl 程序。最初这些工具程序用来显示 Rasmus Lerdorf 的个人履历，以及统计网页流量。他将这些程序和一些窗体解释器集成起来，称为 PHP/FI。PHP/FI 可以和数据库连接，产生简单的动态网页程序。Rasmus Lerdorf 在 1995 年 6 月 8 日将 PHP/FI 公开发布，希望可以通过社区来加速程序开发与查找错误，这个发布的版本命名为 PHP 2，已经是今日 PHP 的一些雏型，它像是类似 Perl 的变量命名方式、窗体处理功能，以及嵌入到 HTML 中运行的能力。程序语法上也类似 Perl，有较多的限制，不过更简单、更有弹性。

在 1997 年，任职于 Technion IIT 公司的两个以色列程序员：Zeev Suraski 和 Andi Gutmans，重写了 PHP 的剖析器，成为 PHP 3 的基础，而 PHP 也在这个时候改称为 PHP：Hypertext Preprocessor。经过几个月测试，开发团队在 1997 年 11 月发布了 PHP/FI 2，随后就开始 PHP 3 的开放测试，最后在 1998 年 6 月正式发布 PHP 3。Zeev Suraski 和 Andi Gutmans 在 PHP 3 发布后开始改写 PHP 的核心，这个在 1999 年发布的剖析器称为 Zend Engine，他们也在以色列的 Ramat Gan 成立了 Zend Technologies 来管理 PHP 的开发。

在 2000 年 5 月 22 日，以 Zend Engine 1.0 为基础的 PHP 4 正式发布，2004 年 7 月 13 日则发布了 PHP 5，PHP 5 则使用了第二代的 Zend Engine[5]。PHP 包含了许多新特色，像是强化的面向对象功能、引入 PDO（PHP Data Objects，一个访问数据库的延伸库），以及许多性能上的增强。目前 PHP 4 已经不会继续更新，以鼓励用户转移到 PHP 5。

2008 年 PHP 5 成为了 PHP 唯一的、正在开发的 PHP 版本。将来的 PHP 5.3 将会加入 Late static binding 和一些其他的功能强化。PHP 6 的开发也正在进行中，主要的改进有移除 register_globals、magic quotes 和 Safe mode 的功能。表 5-1 所示是 PHP 整个发展过程。

表 5-1　PHP 发展历程（来源：http://zh.wikipedia.org/wiki/PHP）（2011-5-16）

主要版本	次要版本	释出日期	说明
1.0	1.0.0	1995 年 6 月 8 日	正式名称为“Personal Home Page Tools (PHP Tools)”，第一次使用了“PHP”的名字
2.0	2.0.0	1996 年 4 月 16 日	针对 PHP 1.0 的改进版，速度更快、体积更小，更容易产生动态网页
3.0	3.0.0	1998 年 6 月 6 日	开发方式改成多人共同参与。Zeev Suraski 和 Andi Gutmans 为了这个版本重写了剖析引擎。
4.0	4.0.0	2000 年 5 月 22 日	改成以 Zend 引擎作为剖析器，具有两阶段剖析/标签剖析系统等先进功能

① http://zh.wikipedia.org/wiki/PHP

续表

主要版本	次要版本	释出日期	说明
4.0	4.1.0	2001年12月10日	加入“超全局变量”(superglobals）功能，包含了$ GET、$ POST、$ SESSION等
	4.2.0	2002年4月22日	默认取消register globals功能。从网络接收的数据将不会设置成全局变量，增加程序安全性
	4.3.0	2002年12月27日	加入命令行可执行文件，称为CLI
	4.4.0	2005年7月11日	Added man pages for phpize and php-config scripts
	4.4.8	2008年1月3日	一些安全性的增强。曾可能为PHP 4的最后版本。若有必要，提供安全性更新到2008-08-08
	4.4.9	2008年8月7日	更多安全性增强和问题修补。PHP 4.4系列的最后版本
5.0	5.0.0	2004年7月13日	Zend Engine II with a new object model
	5.1.0	2005年11月24日	Performance improvements with introduction of compiler variables in re-engineered PHP Engine
	5.2.0	2006年11月2日	默认打开"过滤"的扩展
	5.2.8	2008年12月8日	emergent bug fix
	5.2.9	2009年2月26日	解决了5.2.*的超过了50多个错误和多个安全问题，增加了稳定性
	5.2.10	2009年6月18日	这个版本修正了大量的bug和安全漏洞，并升级了时区数据库
	5.2.17	2011年1月6日	修正了一个浮点数转化的Bug
	5.3.0	2009年6月30日	支持命名空间；使用XMLReader和XMLWriter增强XML支持；支持SOAP，延迟静态绑定，跳转标签（有限的goto），闭包，Native PHP archives
	5.3.3	2010年7月22日	使用命名空间的类中，与类同名的成员函数不再作为构造函数
	5.3.6	2011年3月17日	修正一系列Bug
6.0	6.0.0	时间未定	支持Unicode; 移除ereg扩展，'register_globals', 'magic_quotes'和'safe_mode'; Alternative PHP Cache; Removal of mime_magic and rewrite of fileinfo() for better MIME support

5.2.1.2 PHP特征与功能

PHP在特性方面主要表现在开放的源代码（所有的PHP源代码都可以得到）。PHP是免费的，且具有以下特性：

（1）快捷性好。程序开发快，运行快，技术本身学习快。嵌入HTML：因为PHP可以嵌入HTML语言，它相对于其他语言，编辑简单，实用性强，更适合初学者。

（2）跨平台性强。由于PHP是运行在服务器端的脚本，因此可以运行在UNIX、Linux、Windows下、执行效率高（PHP消耗相当少的系统资源）。

（3）能进行图像处理，即可以用PHP动态创建图像。

（4）面向对象。

在PHP 4、PHP 5中，面向对象方面都有了很大的改进，现在PHP完全可以用来开发大型商业程序。

因此，在技术应用方面主要有伪静态、静态页面生成、数据库缓存、过程缓存、页面处理、大负荷、分布式应用、Jquery 框架集成、Flex、能开发桌面程序应用、连接服务 SDO 数据和支持 MVC 模型等。

而 PHP 的功能可以概述如下①：

（1）别名：在 PHP 4 中，可以利用引用为变量赋值，这给编程带来了很大的灵活性。

（2）扩充了 API 模块：PHP 4.0 为扩展的 API 模块的提供了扩展 PHP 接口模块，它比旧的 API 版本显著地快。PHP 模块已有的和最常用的接口多数被转换到使用这个扩展的接口。

（3）自动资源释放：PHP 4 增加了引用计数功能，这种新技术的引入使 PHP 4 具有了自动内存管理功能，减轻了开发人员的负担。

（4）布尔类型：PHP 4.0 支持布尔类型。

（5）进程生成：在 UNIX 环境下的 PHP 4.0 提供了一个很智能和通用的生成进程，使用了一种名为基于 automake/libtool 的系统生成技术。

（6）COM/DCOM 支持：PHP 4.0 提供 COM/DCOM 支持（仅用于 Windows 环境）可以无缝地存取和访问 COM 对象。

（7）与 PHP 3.0 兼容性很好：PHP 4.0 是与 PHP 3.0 代码向后兼容性接近 100%。由于 PHP 4 的改进的体系结构，两者有一些细微的差别，但是大多数人将可能永远不可能遇上这种情况。

（8）配置：PHP 4 重新设计和增强了 PHP.ini 文件，这使得用 PHP.ini 来配置 PHP 显得极为容易，这个文件可以在运行时被 Apache（unix 系统）或由 Windows 注册（Windows 环境）。

（9）加密支持：PHP 4 实现了完整的加密，这些加密功能是一个完整的 mycrypt 库，并且 PHP 4.0 支持哈希函数、Blowfish、TripleDES 和 MD5。同时，在 PHP 中，SHA1 也是可使用的一些加密算法。

（10）类型检查：PHP 4.0 支持同一操作符用于评类型检查：===（3 等号运算符），为在两个值和其类型之间作检查。例如，3 ===3 将视为假（这是因为类型是不同的），而 3 ==3（当相等判断时）将视为真。

（11）FTP 支持：PHP 4.0 支持 FTP。通常，程序员会为通过一个调制解调器连接下载一个大文件提供一个接口。然而，如果确实有需要，程序员可以使用 PHP。

（12）PHP 4 新增函数或功能增强函数：PHP 4.0 新增了许多函数；同时也将许多现有的函数功能进行了增强，以下是一些例子。array_count_values()、eval()、foreach()、include() ob_end_clean()、ob_end_flush()、ob_get_contents()、 ob_start()、strip_tags()、unset()等。

（13）here 打印：PHP 4.0 的 Here 打印是与 Perl 类似的，尽管完全不相同，但 Here 是打印大容量文章的一个有用的方法，例如在 HTML 文件中，不会漏掉任何一个字符，如目录标记。

（14）HTTP Session fallback 系统：为 HTTP Session 管理的一个 fallback 系统在 PHP 4.0 中被实现。默认情况下，Session 标识符由 cookies 存储。如果没有 cookies 支持或一项 cookies 任务失败，Session 标识符会自动被创建并在 URL 的查询字符串中被携带。

（15）ISAPI 支持：PHP 4.0 能作为一个个性化的 ISAPI 模块作为 IIS 插件。这比 PHP 3.0

① http://baike.baidu.com/view/99.htm

更有效。这时，它作为一个外部的程序来运行。

（16）内存：PHP 4.0能更有效的使用内存，使得较少的内存占用消耗，这主要归功于引用计数技术的实现。

（17）其他类成员函数：在 PHP 4.0 程序员能在成员函数本身的作用域或全局范围内调用其他类的成员函数。例如，程序员能用一个子函数覆盖父函数，并在子函数中调用父函数。

（18）多维数组：在PHP 4.0，利用GET、POST、Cookies的数据传输支持多维数组。

（19）个性化的HTTP Session支持：HTTP Session处理，包括 fallback系统管理，在PHP 4.0被它的新库函数实现。在版本3.0中处理Session要求使用PHPLIB和第三方的库函数，它比把Session直接地由PHP支持慢了许多。

（20）个性化的Java支持：PHP 4.0支持和java的交互。这种个性化的Java支持为PHP在Java对象上创建和使用方法提供一个简单并且有效的工具。

（21）对象和数嵌套组：PHP 4.0实现了功能更加强大的对象，移去了PHP 3.0存在的种种句法限制。同时，对象能在数组以内被嵌套并且反过来也如此，可以根据程序的需要实现嵌套。

（22）面向对象的编程：PHP 4.0为面向对象的编程和构造类，以及对象提供扩展的功能和新特征。PHP4实现了对象重载，引用技术等新技术。

（23）对象重载支持：对象重载语法允许第三方基于面向对象的类库使用PHP4的面向对象的特征存取它们自身的功能。使用这个特征的一个COM模块已经被实现了。

（24）输出缓冲支持：PHP 提供了一个输出缓冲函数集合。输出缓冲支持程序员写包裹函数功能压缩到缓冲区。在PHP4的输出缓冲支持HTML头信息存放，无论HTML的正文是否输出，头信息（header(), content type and cookies）通常不采用缓冲。

（25）增加了PCRE库：PHP 4.0包括一个Perl兼容的正则表达式（PCRE）库，和正常regex库一起与PHP绑定，且split和replace PCRE功能被支持。这样使得PCRE和Perl 正规表达式之间有一些细微差别。

（26）PHP.ini文件：PHP.ini文件在 PHP 4.0被重新设计，使用PHP的配置PHP.ini是更容易并且更有效的。全部文件能被Apache在运行时间操作（在Apache环境下）或由Windows注册表（在 Windows 下面）。被加入 PHP.ini 文件的配置指令自动地在所有相关的模块中被支持。

（27）引用计数：PHP 4.0为系统中的每个数值提供了包括资源的引用计数。一旦一个资源不再被任何变量引用，它自动地被释放以节省内存资源。利用这个特征的最明显的例子就是内置SQL查询的循环语句。而在PHP 3.0中，每次递归另外的SQL结果集合时需要重复申请内存，直到脚本执行完毕，这些结果集合占用的内存才被释放。

（28）支持引用：通过引用可以改变一个变量的值。

（29）函数的运行时绑定：PHP 4.0的运行时间绑定功能允许程序员在他们被声明以前调用，无论声明是否在代码以后或是在运行时间。

（30）类的运行时信息：PHP 4.0支持在运行时刻存取下列类信息：一个对象的类名，一个对象的父类的类名字，以及对象函数所在的名字。

（31）服务器抽象层：为支持 Web 服务器提供了增强型 SAPI（服务器 API）接口，是PHP 4.0不可分的一部分。这个服务器抽象层提供了通用的Web服务器接口支持，支持多线

程 Web 服务器，为大多数的 Web 服务器提供透明的支持。这些服务器包括 Apache、IIS (ISAPI)，以及 AOL 服务器。

（32）语法的点亮显示：PHP 4.0 语法的点亮显示允许开发者看见源代码而不是脚本，这个功能比 PHP 3.0 中的更有效。它跑得更快，执行得更好，并且产生更紧凑的 HTML 代码。

（33）由引用改变变量的值：PHP 4.0 由引用支持可变的赋值，当“关联”的两个变量之中的任何一个的值被改变，另外的变量的值同样被改变，这类似与 C 中的指针类型。

（34）在引用字符串中的变量引用：PHP 4.0 增强了在引用字符串中的变量引用。

（35）在 PHP 中，新增加了构造函数和析构函数、对象的引用和克隆、对象中的私有、公共及受保护模式、接口、命名空间、抽象类、静态成员、异常处理等功能。

下面程序清单 5-1 是 PHP 连接数据库一个基本方法。

程序清单 5-1：

```
...
return $message;
}
function db connect($user='wfuser',
                    $password='wfpass', $db='workflow'){
  mysql connect('localhost', $user, $password)
       or die('I cannot connect to db: ' . mysql error());
}
    foreach ($ POST as $key=>$value) {
       echo "<p>".$key." = " . $value . "</p>";
    }
...
    if (validate($_POST) == "OK") {
        echo "<p>Thank you for registering!</p>";
        db connect();
    } else {
        echo "<p>There was a problem with your registration:</p>";
...
```

5.2.1.3　PHP 基本设计模式

设计模式不仅代表着更快开发健壮软件的有用方法，而且还提供了以友好的术语封装大型理念的方法。设计模式要求在软件研发过程具有松散耦合，也就是说，使系统某个部分中的函数和类不要严重依赖于系统的其他部分函数和类的行为和结构。

1. 工厂模式

工厂模式是一种类，它具有为创建对象的某些方法。可以使用工厂类来创建对象，而不直接使用 new 来完成类的创建。这样，如果想要更改所创建的对象类型，只需更改该工厂即可。使用该工厂的所有代码会自动更改。下面的程序清单 5-2 显示工厂类的一个示例，程序清单 5-3 显示使用工厂方法的一个示例。而在服务器端通常包括两个部分：数据库和一组 PHP 页面，这些页面允许添加反馈、请求反馈列表并获取与特定反馈相关的文章。

程序清单 5-2：

```
<?php
interface IUser
{
  function getName();
}
class User implements IUser
{
  public function __construct( $id ) { }
  public function getName()
  {
    return "Jack";
  }
}
class UserFactory
{
  public static function Create( $id )
  {
    return new User( $id );
  }
}
$uo = UserFactory::Create( 1 );
echo( $uo->getName()."\n" );
?>
```

在程序中，IUser 接口定义用户对象应执行什么操作。IUser 的实现称为 User，UserFactory 工厂类则创建 IUser 对象。

程序清单 5-3：

```
<?php
interface IUser
{
  function getName();
}
class User implements IUser
{
  public static function Load( $id )
  {
      return new User( $id );
  }
  public static function Create( )
  {
      return new User( null );
  }
  public function __construct( $id ) { }
```

```
  public function getName()
  {
    return "Jack";
  }
}
$uo = User::Load( 1 );
echo( $uo->getName()."\n" );
?>
```

在程序中，它仅有一个接口 IUser 和一个实现此接口的 User 类。User 类有两个创建对象的静态方法。

2. 单元素模式

某些应用程序资源是独占的，因为有且只有一个此类型的资源。例如，通过数据库句柄到数据库的连接是独占的。但程序员往往希望在应用程序中共享数据库句柄，因为在保持连接打开或关闭时，它是一种开销，在获取单个页面的过程中更是如此。这时，单元素模式可以满足此要求。如果应用程序每次包含且仅包含一个对象，那么这个对象就是一个单元素（Singleton），见程序清单 5-4 所示。

程序清单 5-4：

```
<?php
require once("DB.php");
class DatabaseConnection
{
  public static function get()
  {
    static $db = null;
    if ( $db == null )
      $db = new DatabaseConnection();
    return $db;
  }
  private $ handle = null;
  private function   construct()
  {
    $dsn = 'mysql://root:password@localhost/photos';
    $this-> handle =& DB::Connect( $dsn, array() );
  }
  public function handle()
  {
    return $this-> handle;
  }
}
print( "Handle = ".DatabaseConnection::get()->handle()."\n" );
print( "Handle = ".DatabaseConnection::get()->handle()."\n" );
?>
```

在程序中显示名为 DatabaseConnection 的单个类。程序员不能创建自己的 Database Connection，因为构造函数是专用的。当使用静态 get 方法时，程序员可以获得且仅获得一个 DatabaseConnection 对象。但是，该方法仅适用于较小的应用程序。在较大的应用程序中，应避免使用全局变量，建议使用对象和方法访问资源。

3．观察者模式

观察者模式提供了避免组件之间紧密耦合的另一种方法，该模式非常简单，即一个对象通过添加一个方法（该方法允许另一个对象，即观察者注册自己）使本身变得可观察。当可观察的对象更改时，它会将消息发送到已注册的观察者。这些观察者使用该信息执行的操作与可观察的对象无关。结果是对象可以相互对话，而不必了解原因，如程序清单 5-5 所示。

程序清单 5-5：

```
<?php
interface IObserver
{
  function onChanged( $sender, $args );
}
interface IObservable
{
  function addObserver( $observer );
}
class UserList implements IObservable
{
  private $ observers = array();
  public function addCustomer( $name )
  {
    foreach( $this-> observers as $obs )
      $obs->onChanged( $this, $name );
  }
  public function addObserver( $observer )
  {
    $this-> observers []= $observer;
  }
}
class UserListLogger implements IObserver
{
  public function onChanged( $sender, $args )
  {
    echo( "'$args' added to user list\n" );
  }
}
$ul = new UserList();
$ul->addObserver( new UserListLogger() );
$ul->addCustomer( "Jack" );
?>
```

在程序中，定义四个元素：两个接口和两个类。IObservable 接口定义可以被观察的对象，UserList 实现该接口，以便将本身注册为可观察。IObserver 列表定义要通过怎样的方法才能成为观察者，UserListLogger 实现 IObserver 接口。该模式不限于内存中的对象，它是在较大的应用程序中使用的数据库驱动的消息查询系统的基础。

4．命令链模式

命令链模式以松散耦合为主，发送消息、命令和请求，或通过一组处理程序发送任意内容，且每个处理程序都会自行判断自己能否处理请求。如果可以，该请求被处理，进程停止。程序员可以为系统添加或移除处理程序，而不影响其他处理程序。为处理请求而创建可扩展的架构时，命令链模式很有价值，使用它可以解决许多问题，如程序清单 5-6 所示。

程序清单 5-6：

```
<?php
interface ICommand
{
  function onCommand( $name, $args );
}
class CommandChain
{
  private $ commands = array();

  public function addCommand( $cmd )
  {
    $this-> commands []= $cmd;
  }
  public function runCommand( $name, $args )
  {
    foreach( $this-> commands as $cmd )
    {
      if ( $cmd->onCommand( $name, $args ) )
        return;
    }
  }
}
class UserCommand implements ICommand
{
  public function onCommand( $name, $args )
  {
    if ( $name != 'addUser' ) return false;
    echo( "UserCommand handling 'addUser'\n" );
    return true;
  }
}
class MailCommand implements ICommand
{
```

```
    public function onCommand( $name, $args )
    {
      if ( $name != 'mail' ) return false;
      echo( "MailCommand handling 'mail'\n" );
      return true;
    }
  }
  $cc = new CommandChain();
  $cc->addCommand( new UserCommand() );
  $cc->addCommand( new MailCommand() );
  $cc->runCommand( 'addUser', null );
  $cc->runCommand( 'mail', null );
  ?>
```

在程序中定义维护 ICommand 对象列表的 CommandChain 类。两个类都可以实现 ICommand 接口：一个对邮件的请求作出响应；另一个对添加用户作出响应。即代码首先创建 CommandChain 对象，并为它添加两个命令对象的实例。然后运行两个命令以查看谁对这些命令作出了响应。如果命令的名称匹配 UserCommand 或 MailCommand，则代码失败，不发生任何操作。

5．策略模式

在此模式中，算法是从复杂类提取的，因而可以方便地替换。例如，如果要更改搜索引擎中排列页的方法，则策略模式是一个不错的选择。思考一下搜索引擎的几个部分：一部分遍历页面，一部分对每页排列；另一部分基于排列的结果排序。在复杂的示例中，这些部分都在同一个类中。通过使用策略模式，可将排列部分放入另一个类中，以便更改页排列的方式，而不影响搜索引擎的其余代码。该模式非常适合复杂数据管理系统或数据处理系统，使得二者在数据筛选、搜索或处理的方式方面需要较高的灵活性，如程序清单 5-7 所示。

程序清单 5-7：

```
  <?php
  interface IStrategy
  {
    function filter( $record );
  }
  class FindAfterStrategy implements IStrategy
  {
    private $ name;
    public function   construct( $name )
    {
      $this-> name = $name;
    }
    public function filter( $record )
    {
      return strcmp( $this > name, $record ) <- 0;
    }
```

```
}
class RandomStrategy implements IStrategy
{
  public function filter( $record )
  {
    return rand( 0, 1 ) > 0.5;
  }
}
class UserList
{
  private $ list = array();
  public function   construct( $names )
  {
    if ( $names != null )
    {
      foreach( $names as $name )
      {
        $this-> list []= $name;
      }
    }
  }
  public function add( $name )
  {
    $this-> list []= $name;
  }
  public function find( $filter )
  {
    $recs = array();
    foreach( $this-> list as $user )
    {
      if ( $filter->filter( $user ) )
        $recs []= $user;
    }
    return $recs;
  }
}
$ul - new UserList( array( "Andy", "Jack", "Lori", "Megan" ) );
$f1   $ul >find( new FindAfterStrategy( "J" ) );
print r( $f1 );
$f2   $ul >find( new RandomStrategy() );
print r( $f2 );
?>
```

在程序中显示了一个用户列表类，它提供了一个根据一组即插即用的策略查找一组用户的方法。UserList 类是打包名称数组的一个包装器。它实现 find 方法，该方法利用几个策略

之一来选择这些名称的子集。这些策略由 IStrategy 接口定义，该接口有两个实现：一个随机选择用户；另一个根据指定名称选择其后的所有名称。

5.2.2 Perl

Perl 是一种脚本语言，于 1987 年 12 月 18 日发表，它借取了 C、sed、awk、shell scripting 及很多其他编程语言的特性。其中最重要的特性是它内部集成了正则表达式的功能，以及巨大的第三方代码库 CPAN。Perl 也是一种高级、通用、直译式、动态的程序语言。最初设计者拉里·沃尔（Larry Wall）为了让在 UNIX 上进行报表处理的工作变得更方便，决定开发一个通用的脚本语言。目前拉里·沃尔已经开发 Perl 6，来作为 Perl 的后继。

Perl 语言直接提供泛型变量、动态数组、Hash 表等更加便捷的编程元素。Perl 具有动态语言的强大灵活的特性，并且还从 C/C++、Basic、Pascal 等语言中分别借鉴了语法规则，从而提供了许多冗余语法。使得程序员可以忽略计算机内部数据存储、类型、处理方法、运算规则、甚至内存越界等等的细节，而将思考中心放在所需要的程序逻辑上。就这一点而言，很多 Perl 程序员认为目前只有 Perl、Python 等泛型语言才能称为高级语言，而 C、Pasca，甚至 C++这些只能称为中高级语言而已。可以说，在统一变量类型和掩盖运算细节方面，Perl 做得比 Python 更为出色。同时，由于 Perl 从其他语言大量借鉴了语法，使得从其他编程语言转到 Perl 语言的程序员可以迅速上手写程序并完成任务，这使得 Perl 语言是一门容易用的语言。但是，Perl 程序的代码令人难以阅读，实现相同功能的程序代码长度可以相差十倍百倍。

5.2.3 Flex

Flex 是最初由 Macromedia 公司在 2004 年 3 月发布的，基于其专有的 Macromedia Flash 平台。并作为富 Internet 应用（RIA）时代的新技术代表（RIA 产生的背景是由于传统网络程序的开发是基于页面的、服务器端数据传递的模式，把网络程序的表现层建立于 HTML 页面之上，而 HTML 是适合于文本的，传统的基于页面的系统已经渐渐不能满足网络浏览者的更高的、全方位的体验要求了）。而在服务器端时，Java 技术提供的功能包括关系型数据库管理系统（RDBM）的连接、服务请求的多线程处理以及随需求增加而进行的最佳伸缩，将这两种技术结合使用可提供一个满足 RIA 应用程序需求的强大的技术堆栈。自从 2007 年 Adobe 公司将其开源以来，Flex 就以前所未有的速度在成长，使很多公司都纷纷加入了 Flex 开发的阵营当中。Flex 代码编译后是一个 SWF 文件，运行在 Flash Player 中，要想看到 SWF 文件在运行时输出的一些调试信息是比较困难的。当开发 Flex 时，一般需要 Java SDK、Tomcat、Eclipse、Flex Builder 和 FireFox。目前，Flex Builder 提供两个版本，一个是 All in one 的版本，另外一个是 Eclipse 的插件版，All in one 的版本内置了一个 Eclipse 的基本核心，插件不全。所以通常采用单独下载 Eclipse 和安装 Flex Builder 插件版的方式。另外在安装过程中不要安装 FlashPlayer 到 IE 或者 FireFox 上。其中，FireFox 可以在扩展组件站点上搜索并安装 Http- Fox，FlashTracer 和 Cache Status 三个插件。

（1）Flex 基本特性。传统的程序员在开发动画应用方面存在困难，Flex 平台最初就是因此而产生。Flex 试图通过提供一个程序员们已经熟知的工作流和编程模型来改善这个问题。

并且 Flex 最初是作为一个 J2EE 应用，或者可以说是 JSP（JavaServer Pages）标签库而发布的。它可以把运行中的 MXML（Flex 用的可扩展置标语言；第一个字母 M 代表 Macromedia 公司；该公司推出了 Flex；该公司于 2005 年被 Adobe 收购）。语言标签中使用 mx 作为前缀，且区分大小写。MXML 是一个强大的声明性 XML 格式。因 XML 格式的声明性质而最大限度地降低构建 GUI 所需的代码量，通过明确分离表示逻辑和交互逻辑降低 GUI 代码的复杂度，在进行软件开发时推进设计模式的使用）和 ActionScript 编译成 FLASH 应用程序（即二进制的 SWF 文件）。最新版的 FLEX 支持创建静态文件，该文件使用解释编译方式，并且不需要购买服务器许可证就可以在线部署。同时，MXML 定义 GUI，而 ActionScript 提供用于处理事件、绑定数据[通过（Bindable）元数据标记]的行为，以及调用远程服务的能力。

Flex 的目标是让程序员更快、更简单地开发 RIA 应用。在多层式开发模型中，Flex 应用属于表现层。同时，Flex 采用 GUI 界面开发，并使用基于 XML 的 MXML 语言。Flex 具有多种组件，可实现 Web Services、远程对象、列排序、图表等功能；Flex 内建动画效果和其他简单互动界面等，相对于基于 HTML 的应用（如 PHP、ASP、JSP、ColdFusion 及 CFMX 等）在每个请求时都需要执行服务器端的模板。由于客户端只需要载入一次，Flex 应用程序的工作流被大大改善。FLEX 的语言和文件结构也试图把应用程序的逻辑从设计中分离出来。

（2）Flex 结构。Flex 提供一个丰富的 UI，支持通过 Adobe Flash®（SWF）应用程序创建、读取、更新和删除（CRUD）联系信息。图 5-1 所示就是基于 Flex 的 Web 应用三层应用程序，其中客户端由嵌入在一个 Web 页面中的 SWF 文件表示，服务器应用程序在一个 Java servlet 容器（本例中为 Apache Tomcat）内运行，且数据库是 MySQL。这三层共同创建一个功能分布式应用程序。

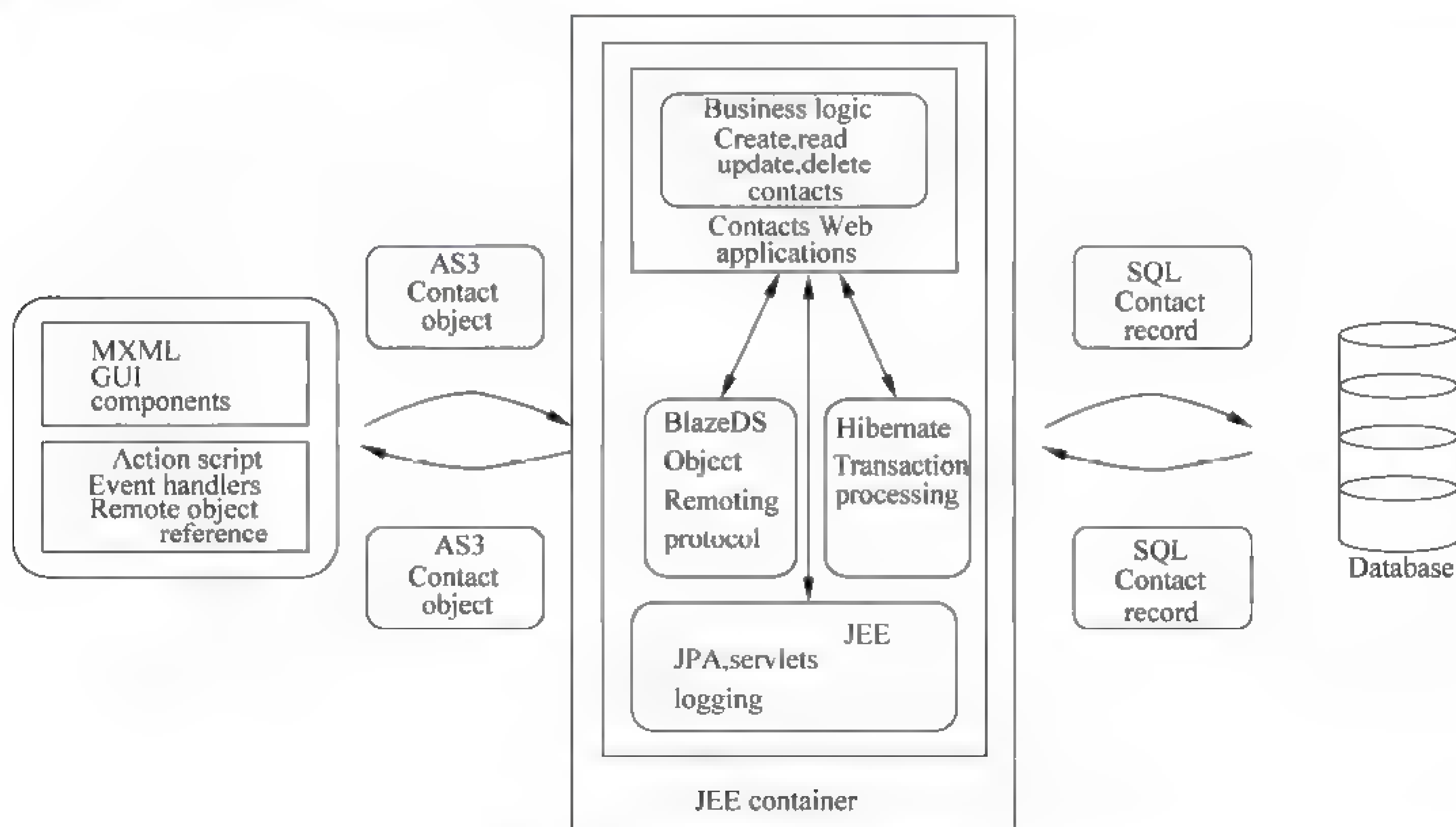

图 5-1　基于 Flex 的 Web 应用程序

在图 5-1 中，Flex 能与 BlazeDS 集成、Flex 4 与 Spring 集成、Flex 4 与 Hibernate 集成。当然 Flex 4 也能与 MVC 集成。

（3）Flex 应用基础。Flex 服务器也是客户端和 Web Services 及远程对象（Coldfusion CFCs，或 Java 类，支持 Action Message Format 的其他对象）之间通讯的通路。一般认为可能是 Flex 替代品的是 OpenLaszlo 和 AJAX 技术。

在 HTML 中嵌入 JavaScript 那样，可以在 mxml 里面嵌入 ActionScript 代码来实现业务逻辑，如程序清单 5-8 是 mxml 的一个“Hello World!”。如果把 Flex 中 mxml 和 ActionScript 的关系理解为 Html 和 JavaScript 的关系，这样就可以很容易理解 Flex 应用方法。ActionScript 语言是面向对象的脚本语言，如程序清单 5-9 就是 ActionScript 程序的例子，它连编写方式都和 JavaScript 非常的相似。除了可以嵌套在 mxml 里面之外，它还可以像 JavaScript 写在单独的.js 文件里面那样写在单独的.as 文件里面，然后在 mxml 里面引入它。

程序清单 5-8：

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"
     initialize="init()">
     <mx:Script>
         <![CDATA[
             private function init():void
             {
                 var i:int = 0;
                 i++;
                 trace("i="+i);
             }
         ]]>
     </mx:Script>
    <mx:Label text="Hello World!" />
</mx:Application>
```

程序清单 5-9：

```
package com.ibm.flex
{
    import flash.events.EventDispatcher;
    import mx.rpc.AsyncToken;
    import mx.rpc.events.FaultEvent;
    import mx.rpc.events.ResultEvent;
    import mx.rpc.http.HTTPService;
    public class J2eeServer extends EventDispatcher
    {
        public function J2eeServer()
        {
        }
        public function sendRequest(locale:String):void
        {
```

```
            var httpObject:HTTPService = new HTTPService();
            httpObject.resultFormat = "text";
            httpObject.url =
                "http://localhost:8080/FlexSample/SampleServlet?locale=
                "+locale;
            var responder:mx.rpc.Responder = new mx.rpc.Responder(onSuccess,
            onFault);
            var call:AsyncToken = httpObject.send();
            call.addResponder(responder);
        }
        private function onSuccess(event:ResultEvent):void
        {
            this.dispatchEvent(event);
        }
             private function onFault(event:FaultEvent):void
        {
            trace("communication failed!");
            this.dispatchEvent(event);
        }
    }
}
```

同时，在这个类定义里面，可以看出 Action Script 编码规范和 Java 非常类似。在程序中定义了一个 sendRequest()方法，使用 HTTPService 对象发起一个 http 的 get 请求，并且对于不同的返回结果定义了 onSuccess()和 onFault()两个方法进行处理。在这两个结果处理方法中，将事件 dispatch 出去。

1．Flex 与 BlazeDS 集成

在实现 Flex 与 BlazeDS 集成时，首先需要在 web.xml 建立一个监听和一个 Servlet。如清单 5-10 所示。

程序清单 5-10：

```
<listener>
        <listener-class>flex.messaging.HttpFlexSession</listener-class>
    </listener>
<servlet>
        <servlet-name>MessageBrokerServlet</servlet-name>
        <servlet-class>flex.messaging.MessageBrokerServlet</servlet-class>
        <init-param>
            <param-name>services.configuration.file</param-name>
            <param-value>/WEB-INF/flex/services-config.xml</param-value>
       </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>MessageBrokerServlet</servlet-name>
```

```
        <url-pattern>/messagebroker/*</url-pattern>
    </servlet-mapping>
...
```

其实将blazeds.war解压到某个目录下，将WEB-INF 下的flex和lib目录复制到 flexProjectForIBM 的 WEB-INF 下面，flex 目录下面包括四个文件，分别是 messageing-config.xml，proxy-config.xml，remoting-config.xml，services-config.xml，建议不要修改这四个文件的名称，若要修改，以及明白具体作用可以参考 BlazeDS 的官方文档。下面以一个接口“HelloWorld”，并新建一个方法 sayHelloToSomeone，并建一个 HelloWorld 的实现类 HelloWorldImpl，实现 sayHelloToSomeone 方法，如程序清单 5-11 所示。

程序清单 5-11：

```
package org.flex.service.impl;
 import org.flex.service.HelloWorld;
 public class HelloWorldImpl implements HelloWorld {
  @Override
  public String sayHelloToSomeone(String name) {
    return "Hello "+name;
  }
 }
```

2. Flex 4 与 Spring 集成

Flex 4 与 Spring 的集成一般有两种方法，一种是通过 annotation 的方式，一种是通过定义 SpringFactory。SpringFactory 继承自 FlexFactory 来实现的，两种方法各有利弊，第一种方法相对简单一些，直接在要暴露的远程服务类上添加@RemotingDestination annotation 就可以了。尤其是多人开发的情况下，不用产生因为多人修改 spring 的配置文件产生冲突而带来的麻烦，当然这种方法他的弊端就是 spring 的 bean 文件与 Flex 有一定的耦合度，给后期的代码维护也会带来一定的困难。第二种方法虽然相对麻烦一些，但是看起来代码显得很清晰，所谓的麻烦也只是在当初配置的时候而已，配置完之后，进入正常开发中就是第一种方法一样方便了。所以下面以第二种的集成方法为例进行说明。首先将下载的 Spring Framework 的相关 JAR 文件放到的 web 工程的 lib 下面，然后添加 applicationContext.xml 文件，添加 HelloWorldImpl 类到该文件中，applicationContext.xml 文件内容见程序清单 5-12，并且需要在 services-config.xml 中添加 Factory（<factories> <factory id="spring" class="org.flex.spring.SpringFactory"/> </factories>）。如程序清单 5-13 是在 Web.xml 添加对 Spring 支持。

程序清单 5-12：

```
<beans xmlns="http://www.springframework.org/schema/security"
    xmlns:b="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation=
    "http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/security
```

```
    http://www.springframework.org/schema/security/spring security 3.0.xsd">
   <b:bean id "helloWorld" class "org.flex.service.impl.HelloWorldImpl">
   </b:bean>
 <beans>
```

程序清单 5-13：

```
<listener>
       <listener-class>org.springframework.web.context.ContextLoaderListener
         </listener-class>
   </listener>
     <context-param>
         <param-name>contextConfigLocation</param-name>
 <param-value>/WEB-INF/applicationContext.xml</param-value>
     </context-param>
     <servlet>
         <servlet-name>SpringLog4jConfigServlet</servlet-name>
         <servlet-class>org.springframework.web.util.Log4jConfigServlet
         </servlet-class>
     </servlet>
     <servlet>
         <servlet-name>web</servlet-name>
         <servlet-class>org.springframework.web.servlet.DispatcherServlet
         </servlet-class>
     </servlet>
```

这时就可以用 SpringFactory 来添加的实现 FlexFactory 的类了，如程序清单 5-14 是 Spring Factory 类的代码清单。

程序清单 5-14：

```
package org.flex.spring;
 import org.springframework.context.ApplicationContext;
 import org.springframework.web.context.support.WebApplicationContextUtils;
 import org.springframework.beans.BeansException;
 import org.springframework.beans.factory.NoSuchBeanDefinitionException;
 import flex.messaging.FactoryInstance;
 import flex.messaging.FlexFactory;
 import flex.messaging.config.ConfigMap;
 import flex.messaging.services.ServiceException;
 public class SpringFactory implements FlexFactory {
  private static final String SOURCE = "source";
  public void initialize(String id, ConfigMap configMap) {}
  public FactoryInstance createFactoryInstance(String id, ConfigMap properties)
  {
      SpringFactoryInstance instance = new SpringFactoryInstance(this, id,
      properties);
```

```
        instance.setSource(properties.getPropertyAsString(SOURCE, instance
        .getId()));
        return instance;
    }


    public Object lookup(FactoryInstance inst)
    {
        SpringFactoryInstance factoryInstance = (SpringFactoryInstance) inst;
        return factoryInstance.lookup();
    }
    static class SpringFactoryInstance extends FactoryInstance
    {
        SpringFactoryInstance(SpringFactory factory, String id, ConfigMap
        properties)
        { super(factory, id, properties);
        }
        public String toString()
        {
            return "SpringFactory instance for id=" + getId() + \
            " source=" + getSource() + " scope=" + getScope();
        }
  public Object lookup()
   {ApplicationContext appContext = WebApplicationContextUtils.getWebApplication
   Context(
        flex.messaging.FlexContext.getServletConfig().getServletContext());
            String beanName = getSource();
  try{return appContext.getBean(beanName);}
            catch (NoSuchBeanDefinitionException nexc)
            {ServiceException e = new ServiceException();
                String msg = "Spring service named '" + beanName +
                "' does not exist.";e.setMessage(msg);e.setRootCause(nexc);
                e.setDetails(msg);e.setCode("Server.Processing");throw e;}catch (
                  BeansException bexc){
                ServiceException e = new ServiceException();
                String msg = "Unable to create Spring service named '" +
                beanName + "' ";e.setMessage(msg);e.setRootCause(bexc);e.set
                Details(msg);
   e.setCode("Server.Processing");throw e;
            }
        }
    }
}
```

3．Flex 4 与 Hibernate 集成

在实现 Flex 4 与 Hibernate 集成时，需要配置 hibernate.cfg.xml，如程序清单 5-15 所示。

然后需要加入 sessionFactory 实现事务管理，如程序清单 5-16 所示，其实真正需要集成的是 Spring 与 Hibernate，这在后续章节进行详解。

程序清单 5-15：

```
<hibernate-configuration>
  <session-factory name="sessionFactory">
  <property name="
  hibernate.connection.driver class">oracle.jdbc.driver.OracleDriver</property>
  <property name="
  hibernate.connection.password">anyone</property>
  <property name="
  hibernate.connection.url">jdbc:oracle:thin:@localhost:orcl</property>
  <property name="
  hibernate.connection.username"> anyone </property>
  <property name="
  hibernate.dialect">org.hibernate.dialect.OracleDialect</property>
    </session-factory>
 </hibernate-configuration>
```

程序清单 5-16：

```
<b:bean id="sessionFactory"
  class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
  <b:property name="configLocations" value="classpath:hibernate.cfg.xml" />
     </b:bean>
```

5.2.4　Harmony

Apache Harmony 的提案在 2005 年 5 月被 Apache 软件基金会（ASF）接受，并且按照 ASF 惯例成为一个孵化器（incubator）项目。它为自己定了两个目标，首先是开发出一个独立并且与现有 JDK 兼容的 Java SE 5 实现，并且以 Apache 软件许可证 2.0 版发行。其次是建立一个开放的模块化运行时架构，包括虚拟机和类库之间及其内部的模块化，并通过这个平台，允许社区在此基础上自由定制自己的 Java 实现，或者对某个模块单独进行创新。

Apache Harmony 项目的成立以及它的这两个目标具有很大的现实意义。首先，由于商业 JDK 的流行性，它们几乎成为事实上的标准，所以 Harmony 必须与它们保持高度的兼容，才能够使应用程序的迁移成本最低，也就相对容易被用户所接受。其次，Harmony 存在的重要意义之一就在于这是一个属于开源社区的 Java 平台，在这个平台上，社区可以自由的移植和创新，而一个开放的模块化的架构，将为移植和创新带来最大的便利性。最后，Apache 软件许可证是一个对商业公司和开源社区都比较友好的开源许可证，因此 Harmony 可以给最大范围的开发人员和用户带来便利。图 5-2 所示是 Harmony 模块化的结构。

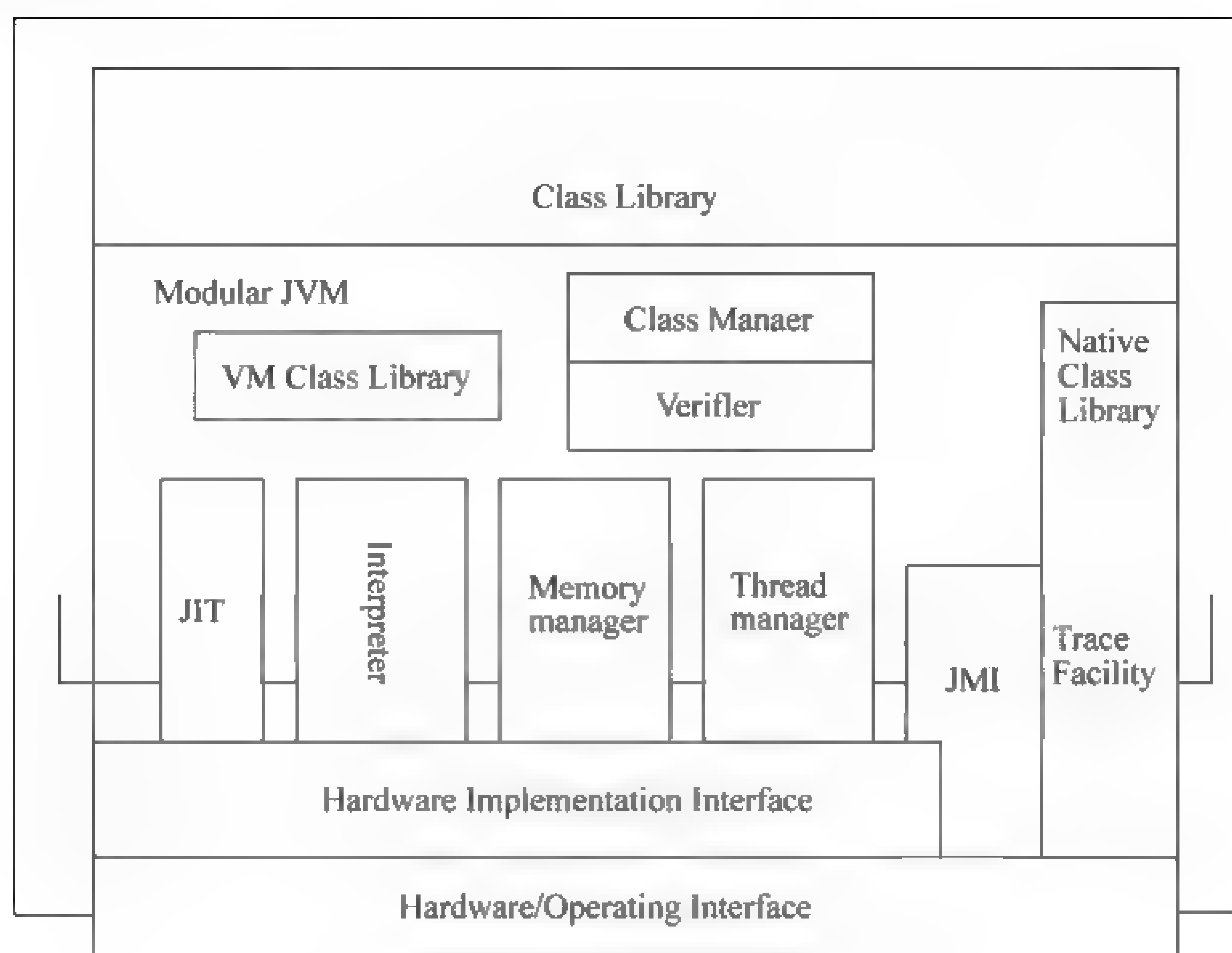

图 5-2　Harmony 模块化的结构

目前，Apache Harmony 已经拥有了一个活跃的开发社区，并且接受了来自公司，学校和个人的多次捐赠。此外 Harmony 项目还接受了三个 Java 虚拟机实现的捐赠，另外还有一个开源 Java 虚拟机 SableVM 正在积极的与 Harmony 社区合作以实现与 Harmony 类库的集成。而 Harmony 与 Java SE 实现的难度最大的是其规模庞大的类库，而最近的类库 API 覆盖率统计表明，Harmony 的 J2SE 1.4.2 类库 API 的覆盖率已经超过 80%，Java SE 5 的覆盖率已经达到 79%以上。目前经不完全测试，在 Harmony 上已经可以良好运行 Eclipse、JEdit、Ant 等流行的 Java 工具，并且可以部分运行 Tomcat 和 Geronimo 等企业应用。不过有时在 Eclipse 下进行 Harmony 的开发还需要下载其他的一些组件，如 Subclipse（集成在 Eclipse 上的 Subversion 客户端）、Snapshot（编译构建好的可执行快照）、Source（类库最新的源码，可用"svn co address"检出到本地）、Virtual Machine、VM plugin（让 Eclipse 识别模块化的 Harmony JRE 的插件）。并且 Harmony 的一个重要目标是类库的模块化，并将进一步采用 OSGi 运行时框架技术，根据模块元数据描述，自动装载模块，

下面对 Harmony 的 Port Layer，VMI 和 Java 统一启动程序作进一步的介绍：

5.2.4.1　Apache Harmony Port Layer

Port Layer 是位于操作系统 API 和应用程序之间的函数库，它是由一个标准 C 的库（Port Library）来实现得。Port Library 与操作系统交互，为虚拟机和类库的本地代码提供了一个平台无关的标准 C 语言的 API 来访问系统调用，诸如文件 I/O、网络 I/O、内存操作、信号处理，以及错误处理等功能都被纳入 Port Library 的范围。在 Port Layer 内部，不同平台拥有 Port Layer 的不同实现，在这些实现中使用了实际平台的系统调用。而对外，Port Layer 提供了一套统一的 API，屏蔽了底层系统的差异，从而使得开发者既不必关心软件究竟在何种平台上开发，也不用关心在何种平台上运行。在当今的软件世界里，许多需要在不同平台上开发、运行的软件，比如 HTTP 服务器，数据库等，都可能需要 Port Layer。比如 Apache Http Server，

就使用了 APR（Apache Portable Runtime），使它在不同平台上都可以运行。

5.2.4.2　Apache Harmony 虚拟机接口

Harmony 为了模块化和可移植性的要求，定义了虚拟机接口（VMI）。VMI 的存在，使 Harmony 的 Class Lib 和虚拟机的分离成为了可能。只要实现了该接口的虚拟机，就可以和 Harmony 的类库兼容，也可以与 Harmony 的类库实现互操作，并且可以与 Harmony 提供的 Java 启动程序（launcher）协作。

Harmony 虚拟机接口分为三部分，第一部分是 Java 语言接口，由 23 个内核类(Kernel Class)组成；第二部分是 C 语言接口，由 10 个函数组成，第三部分就是标准 JNI(Java Native Interface)。分别介绍如下：

（1）内核类的提出是因为少数核心的公共类是与虚拟机密切相关（VM-specific）的，它们都是属于 java.lang，java.lang.ref，java.lang.reflect 和 java.security 等几个核心的包，比如说 java.lang.ClassLoader，java.lang.ref.WeakReference 等。随着 Java 版本的升级，核心类的数量也可能会增加。同时，Harmony 的类库实现为大多数核心类定义了实现模板。使得虚拟机的开发人员可以从零开始实现这些核心类，也可以在 Harmony 提供的模板基础上开始开发。

（2）虚拟机接口还定义了 VM 必须实现的 10 个 C 函数，用来访问虚拟机和类库共享的数据结构和接口，比如说访问操作系统抽象库（Port Library），虚拟机本地存储空间等。

（3）虚拟机的最后一部分就是标准 JNI，它在这里的主要用途是允许从 Java 类库的本地链接库中创建 Java 对象。

5.2.4.3　Apache Harmony 类库的模块化

Apache Harmony 将类库分成易于管理的模块。类库的模块实际上是由一组实现相关的功能的 Java 类和本地程序组成的，目前它们按照包的级别来划分，也就是说不会有两个属于同一个 Java package 的类出现在两个不同的模块里。这些模块可以包含私有的内部实现，但是必须导出 Java API 参考文档规定标准 API，也可以导出 Harmony 独有的 API，但是这些 API 并不鼓励用户使用，只是供其他模块调用的。

模块的鉴别方法是：首先根据 Java 规范，确定各 Java 包之间的依赖关系；然后寻找弱耦合的部分，将本身内聚的包组织在一起，最后将在社区讨论哪部分类库拆开独立成一个模块。目前 Harmony 将 Java SE 5 规定的类库分成了 31 个模块（ACCESSIBILITY、ANNOTATION、APPLET、ARCHIVE、AUTH、AWT、BEANS、CONCURRENT、CRYPTO、IMAGEIO、INSTRUMENT、JMX、JNDI、LANG-MANAGEMENT、LOGGING、LUNI （lang，util，net，io）、MATH、NIO-CHANNELS、NIO-CHARSET、ORB、PREFS、PRINT、REGEX、RMI、SECURITY、SOUND、SQL、SWING、TEXT、X-NET、XML），在 Java 升级之后，模块的数量可能会增加。

在 Harmony 的代码资源库中，每个模块的源文件都保存在自己的各自的路径中，如果使用 Eclipse 做开发工具，每个模块就可以看作一个单独的工程项目。特别需要注意的是，按照 jar 文件的规范，在每个模块都有一个 META-INF/manifest.mf 文件，在这个文件中定义了该模块的 OSGI 元数据，其中最重要的就是该模块导入导出的 Java package。如程序清单 5-17。

程序清单 5-17：

```
Manifest Version: 1.0
 Bundle ManifestVersion: 2
 Bundle Name: Harmony NIO
 Bundle-SymbolicName: org.apache.harmony.nio
 Bundle-Version: 1.0.0
 Bundle-ClassPath: .
 Eclipse-JREBundle: true
 Import-Package: java.io,
 java.lang,
 java.lang.ref,
 java.lang.reflect,
 java.net,
 java.nio.charset,
 java.security,
 java.util,
 org.apache.harmony.kernel.vm,
 org.apache.harmony.luni.net,
 org.apache.harmony.luni.platform,
 org.apache.harmony.luni.util,
 tests.support;hy usage=test;resolution:=optional,
 tests.util;hy usage=test;resolution:=optional
 Export-Package: java.nio,
 java.nio.channels,
 java.nio.channels.spi,
 org.apache.harmony.nio
```

5.2.4.4 Apache Harmony 启动程序

Apache Harmony 提供了一个启动程序，也就是通常用来运行 Java 程序的 java 和 javaw 命令的实现，这个启动程序是个多目标的启动器，可以根据命令行参数启动不同的虚拟机，也可以支持虚拟机特有的选项，如属性文件或命令行参数等。这一切都是基于虚拟机接口、Port Library 和标准 JNI 实现的。

5.2.5 JSP

JSP（JavaServer Pages）是由 Sun Microsystems 公司倡导和许多公司参与共同创建的一种使软件开发者可以响应客户端请求，而动态生成 HTML、XML 或其他格式文档的 Web 网页的技术标准。JSP 技术是以 Java 语言作为脚本语言的，JSP 网页为整个服务器端的 Java 库单元提供了一个接口来服务于 HTTP 的应用程序。它被 JSP 编译器编译成 Java Servlets。一个 JSP 编译器可以把 JSP 编译成 JAVA 代码写的 servlet 然后再由 JAVA 编译器来编译成机器码，也可以直接编译成二进制码。

JSP 使 Java 代码和特定的预定义动作可以嵌入到静态页面中。JSP 句法增加了被称为 JSP 动作的 XML 标签，它们用来调用内建功能。另外，还可以创建 JSP 标签库，然后像使用

标准 HTML 或 XML 标签一样使用它们。而标签库提供了一种与平台无关的扩展服务器性能的方法。

从架构上说，JSP 可以被看作是从 Servlets 高级提炼，并作为 JAVA Servlet 2.1 API 的扩展而应用。Servlets 和 JSPs 最早都是由 Sun Microsystems 开发的。从 JSP1.2 版本以来，JSP 处于 Java Community Process（有人译为 JAVA 社区组织）开发模式下。同时，JSR-53 规定了 JSP 1.2 和 Servlet 2.4 的规范，JSR-152 规定了 JSP 2.0 的规范。2006 年 5 月，JSP 2.1 的规范作为 Java EE 5 的一部分，在 JSR-245 中发布。

从功能上讲，JSP 具有将内容的生成和显示进行分离，实现可重用组件；并采用让大家都明白的标识来实现跨平台和连接多种数据库等特性。

JSP 像 PHP 等网页语言一样，网页页面一般具有诸如 HTML 静态数据和 JSP 指令等，如 include 指令、JSP 动作和用户自定义标签。在实现动态交互时，还需要使用 JSP 对象：request 对象、response 对象、session 对象、application 对象、out 对象、page java.lang.Object、config、exception、pageContext。下面就选择几个典型代表进行进一步说明：

（1）静态数据。静态数据是指在输入文件中的内容和输出给 HTTP 响应的内容完全一致。此时，该 JSP 输入文件会是一个没有内嵌 JAVA 或动作的 HTML 页面。而且，客户端每次请求都会得到相同的响应内容。

（2）JSP 指令。JSP 指令控制 JSP 编译器生成 servlet：

① 包含指令：包含指令 include 通知 JSP 编译器把另外一个文件完全包含入当前文件中（如<%@ include file="somefile.jsp" %>）。效果就好像被包含文件的内容直接被粘贴到当前文件中一样，这个功能和 C 预处理器所提供的很类似。

② 页面指令 page。import 使一个 JAVA 导入声明被插入到最终页面文件，例如：

```
<%@ page import="java.util.*" %>
```

表示导入样例。

注：在同一个 JSP 文件中只有“import”导入页面指令可以被多次使用。

contentType 规定了生成内容的类型。即当生成非 HTML 内容或者当前字符集 character set 时，并非默认字符集时使用。例如：

```
<%@ page contentType="text/html" %>
```

表示页面类型。

errorPage 处理 HTTP 请求时，如果出现异常则显示该错误提示信息页面。

例如：

```
<%@ page isErrorPage=false %>
```

表示无无错页面。

isErrorPage 如果设置为 TRUE，则表示当前文件是一个错误提示页面。

isThreadSafe 表示最终生成的 servlet 是否安全线程（thread safe）。

例如：

```
<%@ page isThreadSafe-true %>
```

表示 JSP 的安全线程。

③ 标签库指令 taglib。标签库指令描述了要使用的 JSP 标签库。该指令需要指定一个前

缀 prefix（和 C++的命名空间很类似）和标签库的描述 URI。例如：

```
<%@ taglib prefix="myprefix" uri="taglib/mytag.tld" %>
```

（3）JSP 动作。JSP 动作是一系列可以调用内建于网络服务器中的功能的 XML 标签。JSP 主要提供了以下动作：

jsp:include 和子过程类似，JAVA SERVLET 暂时接管对其他指定的 JSP 页的请求和响应。当处理完该 JSP 页后就马上把控制权交还当前 JSP 页。这样 JSP 代码就可以在多个 JSP 页中共享而不用复制。例如：

```
<html>
<head></head>
<body>
<jsp:include page="mycommon.jsp" >
   <jsp:param name="extraparam" value="myvalue" />
</jsp:include>
name:<%=request.getParameter("extraparam")%>
</body>
</html>
```

jsp:param 可以在 jsp:include，jsp:forward 或 jsp:params 块之间使用。指定一个将加入请求的当前参数组中的参数。

jsp:forward 用于处理对另一个 JSP 或 SERVLET 的请求和响应。控制权永远不会交还给当前 JSP 页。例如：

```
<jsp:forward page="subpage.jsp" >
    <jsp:param name="forwardedFrom" value="this.jsp" />
</jsp:forward>
```

jsp:plugin Netscape Navigator 的老版本和 Internet Explorer 使用不同的标签以嵌入一个 applet。这个动作产生为嵌入一个 APPLET 所需要的指定浏览器标签。例如：

```
<jsp:plugin type=applet height="100%" width="100%"
          archive="myjarfile.jar,myotherjar.jar"
          codebase="/applets"
          code="com.foo.MyApplet" >
   <jsp:params>
      <jsp:param name="enableDebug" value="true" />
   </jsp:params>
   <jsp:fallback>
      Your browser does not support applets.
   </jsp:fallback>
</jsp:plugin>
```

jsp:fallback 如果浏览器不支持 APPLETS 则会显示的内容。

jsp:getProperty 从指定的 JavaBean 中获取一个属性值。

jsp:setProperty 在指定的 JavaBean 中设置一个属性值。

jsp:useBean 创建或者复用一个 JavaBean 变量到 JSP 页。例如：

```
<jsp:useBean id="myBean" class="com.foo.MyBean" scope="request" />
<jsp:getProperty name="myBean" property="lastChanged" />
<jsp:setProperty name="myBean" property="lastChanged" value="<%= new Date()%>" />
```

其中，scope 属性可以是 request，page，session，application。

（4）JSP 标签库。除了 JSP 预定义动作之外，开发者还可以使用 JSP 标签扩展 API 来添加他们自定义的动作。开发者可以实现一个标签的界面和一个标签库的 XML 描述文件的 JAVA 类，这就能指定标签和实现标签的 JAVA 类，例如下面的 JSP：

```
<%@ taglib uri="mytaglib.tld" prefix="myprefix" %>
...
<myprefix:myaction> <%-- the start tag %>
...
</myprefix:myaction> <%-- the end tag %>
...
public class MyActionTag extends  TagSupport {
    public void release() {...}
    public MyActionTag() { ... }
     //called for the start tag
    public int doStartTag() { ... }
     //called at the end tag
}
```

（5）request 对象。该对象封装了用户提交的信息，并通过调用该对象相应的方法可以获取封装的信息，它是 HttpServletRequest 的实例。request 对象可以获得客户端的输入信息，且包括了从客户端传来的请求信息。在 HTTP 1.1 协议中，客户端请求信息是从客户端通过 HTTP 头（HTTP Header）和消息体传送到服务器端的。具体详细讲解可参照：http://ajava.org/readbook/java/rhzgf/4912.html。

（6）session 对象。session 对象是十分重要的一个 JSP 内置对象，它可以用来在每一个用户之间分别保存用户信息，这与 application 对象不同。application 对象用于在多个程序之间保存信息，application 对象只有一个，但它可以绑定若干个相当于全局变量的参数或者 Java 对象，每个 JSP 程序所访问的都是 application 对象的一个都是一样同步副本，而且 application 对象的生命周期贯穿服务器的整个运行周期。但是，服务器上的 session 对象却可以有多个，不同的用户所面临的 session 对象一般来说是不同的，当用户登录网站，系统将为它生成一个独一无二的 session 对象，用以记录该用户的个人信息，一旦该用户退出网站，那么该 session 对象将会被注销。session 对象也可以绑定若干个参数或者 Java 对象，这些参数或者 Java 对象就相当于局部变量，且不同 session 对象间的同名变量是不会相互干扰的。

同时，session 对象其实是 javax.servlet.http.HttpSession 接口的实例对象。因此，session 对象的方法其实就是 HttpSession 接口的方法。

5.2.6　Android

Android 是一种基于 Linux® V2.6 内核的综合操作环境。最初，Android 的部署目标是移动电话领域，包括智能电话和更廉价的翻盖手机。但是，Android 全面的计算服务和丰富的功能支持完全有能力扩展到移动电话市场以外。Android 也可以用于其他的平台和应用程

序。Android 还是一个年轻的、有待开发的平台，它有潜力同时涵盖移动电话的两个不同消费群体，甚至可能缩小工作和娱乐之间的差别。

Android 平台是 Open Handset Alliance 的成果，Open Handset Alliance 组织由一群共同致力于构建更好的移动电话的公司组成。这个组织由 Google 领导，包括移动运营商、手持设备制造商、零部件制造商、软件解决方案和平台提供商以及市场营销公司。从软件开发的观点看，Android 正处在开源领域的中心位置。市场上第一款支持 Android 的手机是由 HTC 制造并由 T-Mobile 供应的 G1。这款设备从设想到推出花了大约一年的时间，唯一可用的软件开发工具是一些实行增量改进的 SDK 发行版。随着 G1 发行日的临近，Android 团队发布了 SDK V1.0，是一个 ZIP 包，它通常在 Eclipse 下使用，这个包主要包括：

（1）android.jar：Java 归档文件，其中包含构建应用程序所需的所有 Android SDK 类。

（2）documention.html 和 docs 目录：本地和网上提供的 SDK 文档。这些文档的主要形式为 JavaDocs，以便于在 SDK 中导航大量的包。文档还包括一个高级开发指南和 Android 社区的链接。

（3）Samples 目录：samples 子目录包含各种应用程序的源代码，包括 ApiDemo，它演示了很多 API。这个示例应用程序可以作为 Android 应用程序开发的良好起点。

（4）Tools 目录：包含所有用于构建 Android 应用程序的命令行工具。其中最常用、最有用的工具是 adb 实用程序（Android Debug Bridge）。

（5）usb_driver：该目录包含将开发环境连接到支持 Android 的设备（例如 G1 或 Android Dev 1 解锁开发手机）所需的驱动程序。只有 Windows 平台的开发人员才需要这些文件。

Android 下载地址是：http://developer.android.com/index.html。同时，为了鼓励创新，Google 举办了两届“Android Developer Challenges”，为优胜的参赛作品提供数百万美金的奖励。G1 问世几个月之后，随后就发布了 Android Market，它使用户可以浏览应用程序，并且可以将应用程序直接下载到他们的手机上。

1. Android 平台

Android 有丰富的功能，因此很容易与桌面操作系统混淆。Android 是一个分层的环境，构建在 Linux 内核的基础上，如图 5-3 所示。而 Android 的 UI 子系统包括窗口、视图、用于显示一些常见组件（例如编辑框、列表和下拉列表）的小部件等。

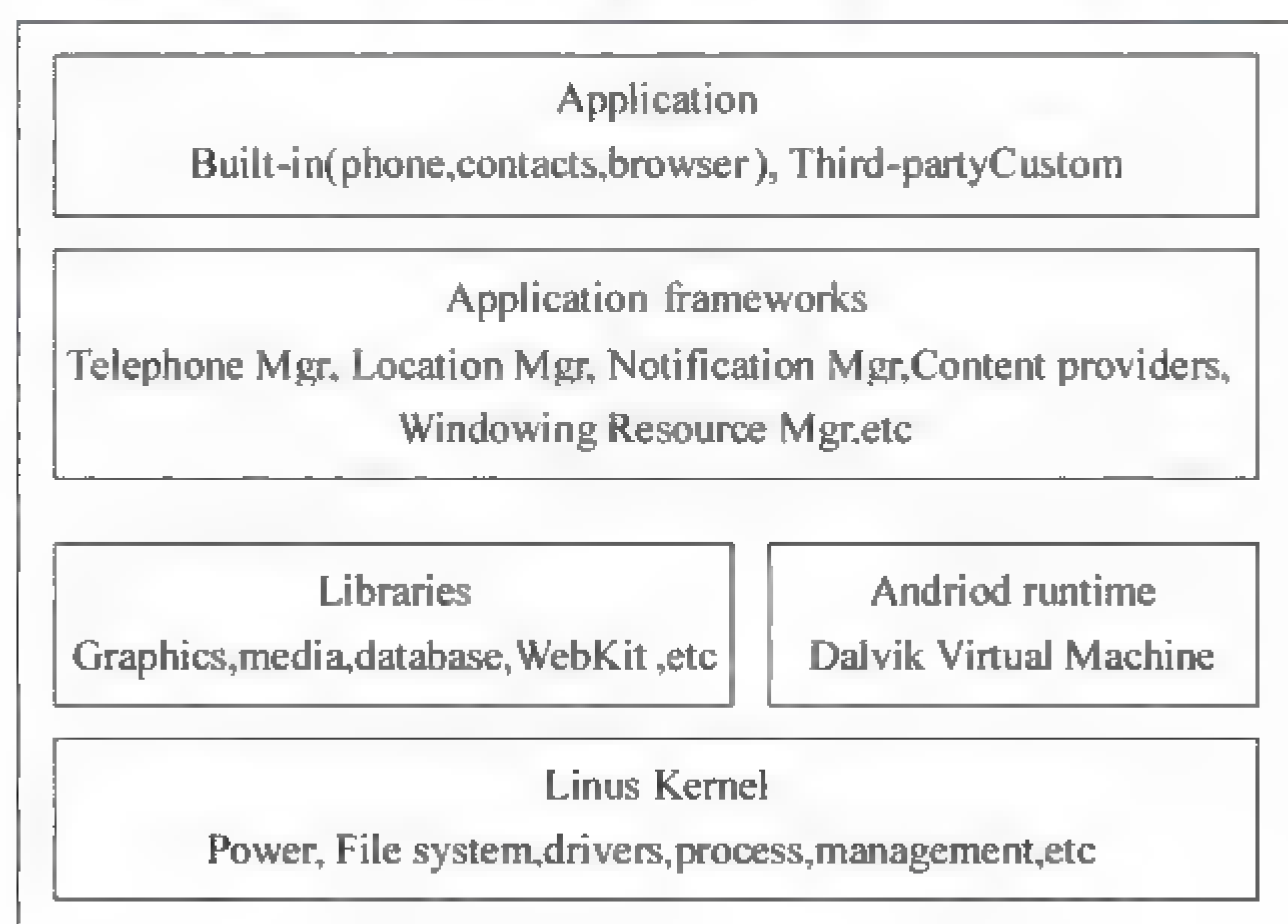

图 5-3　Android 层次结构

Android 包括一个构建在 WebKit 基础上的可嵌入浏览器，iPhone 的 Mobile Safari 浏览器

同样也是以 WebKit 为基础。

Android 提供多种连接选项，包括 WiFi、蓝牙和通过蜂窝（cellular）连接的无线数据传输（例如 GPRS、EDGE 和 3G）。Android 应用程序中一项流行的技术是链接到 Google 地图，以便在应用程序中显示地址。Android 软件栈还提供对基于位置的服务（例如 GPS）和加速计的支持，不过并不是所有的 Android 设备都配备了必需的硬件，另外 Android 还有摄像支持。

过去，移动应用程序努力向桌面应用程序看齐的两个领域分别是图形/媒体和数据存储方法。Android 通过提供对 2D 和 3D 图形的内置支持，包括 OpenGL 库，解决了图形方面的挑战。由于 Android 平台包括流行的开源 SQLite 数据库，因此缓解了数据存储的负担。

2．Android 应用程序结构

Android 运行在 Linux 内核上。Android 应用程序是用 Java 编程语言编写的，它们在一个虚拟机（VM）中运行。但这个 VM 并非想象中的 JVM，而是 Dalvik Virtual Machine，它也是一种开源技术。每个 Android 应用程序都在 Dalvik VM 的一个实例中运行，如图 5-4 所示，这个实例驻留在一个由 Linux 内核管理的进程中。

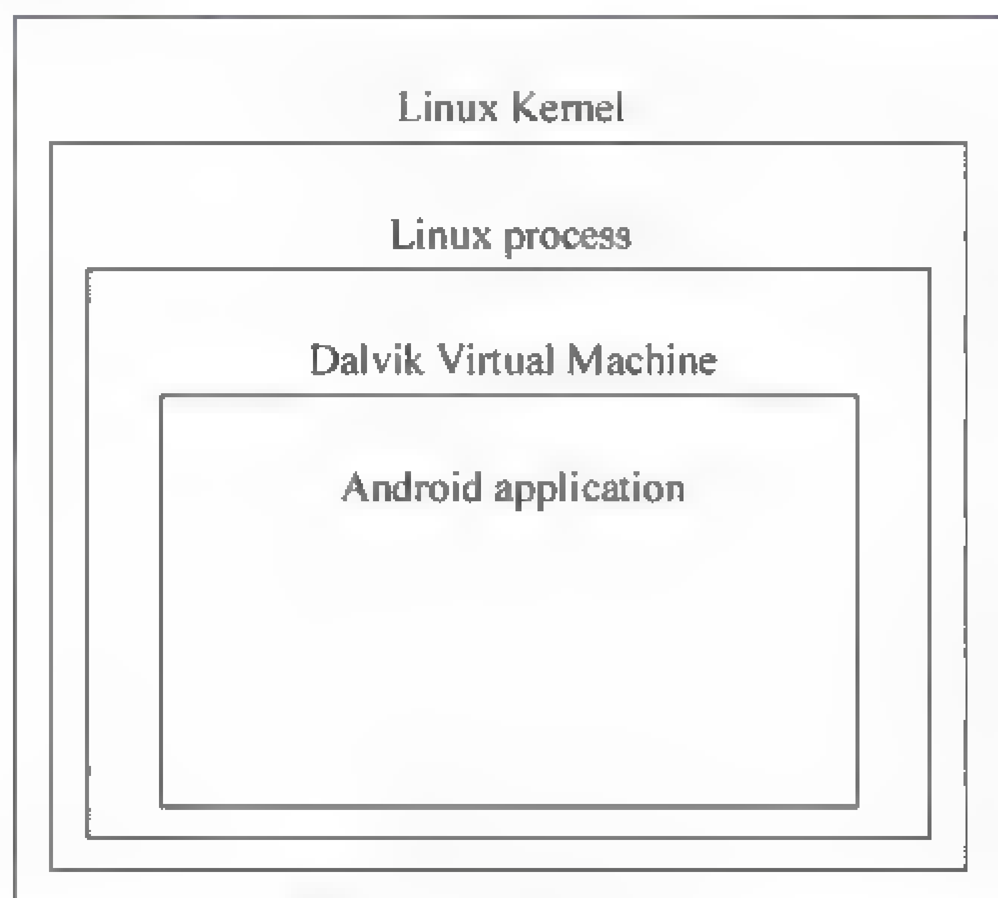

图 5-4　Dalvik VM

Android 应用程序是连同一个 AndroidManifest.xml 文件一起部署到设备的。AndroidManifest.xml 包含必要的配置信息，以使将它适当地安装到设备。它也包括必需的类名和应用程序能够处理的事件类型，以及运行应用程序所需的许可。通常 Android 应用程序由一个或多个组件组成。

（1）活动。具有可视 UI 的应用程序是用活动实现的。当用户从主屏幕或应用程序启动器选择一个应用程序时，就会开始一个动作。

（2）服务。服务应该用于任何需要持续较长时间的应用程序，例如网络监视器或更新检查应用程序。

（3）内容提供程序。可以将内容提供程序看作数据库服务器。内容提供程序的任务是管理对持久数据的访问，例如 SQLite 数据库。如果应用程序非常简单，那么可能不需要创建内容提供程序。如果要构建一个较大的应用程序，或者构建需要为多个活动或应用程序提供数据的应用程序，那么可以使用内容提供程序实现数据访问。

（4）广播接收器。Android 应用程序可用于处理一个数据元素，或者对一个事件（例如接收文本消息）做出响应。

5.3 常用的开发环境

下面介绍开源软件的开发环境，这是因为很多开源软件不能独立运行，需要依托一个开源软件环境作为运行支撑。其中Eclipse几乎是所有基于Java、PHP等开源软件的运行环境的基础，即将开源软件的JAR包括部署到Eclipse中进行应用。同时Eclipse也是多款开源软件集成的重要支撑平台，即插件的安装通常是通过解压缩下载文件，并将其内容复制到Eclipse插件目录来完成的。而目前以Eclipse为基础的商业化软件myEclipse已经得了广泛了使用。下面针对Eclipse、CVS、NetBeans、Apache Ant和Junit进行概述和总结分析。

5.3.1 Eclipse

Eclipse是著名的跨平台的自由集成开发环境（Integrated Development Environment，IDE）。最初主要用来Java语言开发，目前有人通过插件使其作为C++、Python、PHP等其他语言的开发工具。它本身只是一个框架平台，但是众多插件的支持，使得Eclipse拥有较佳的灵活性。目前，许多软件开发商以Eclipse为框架开发自己的IDE，它满足开源软件的Eclipse通用公共许可证，图5-5是Eclipse结构图。

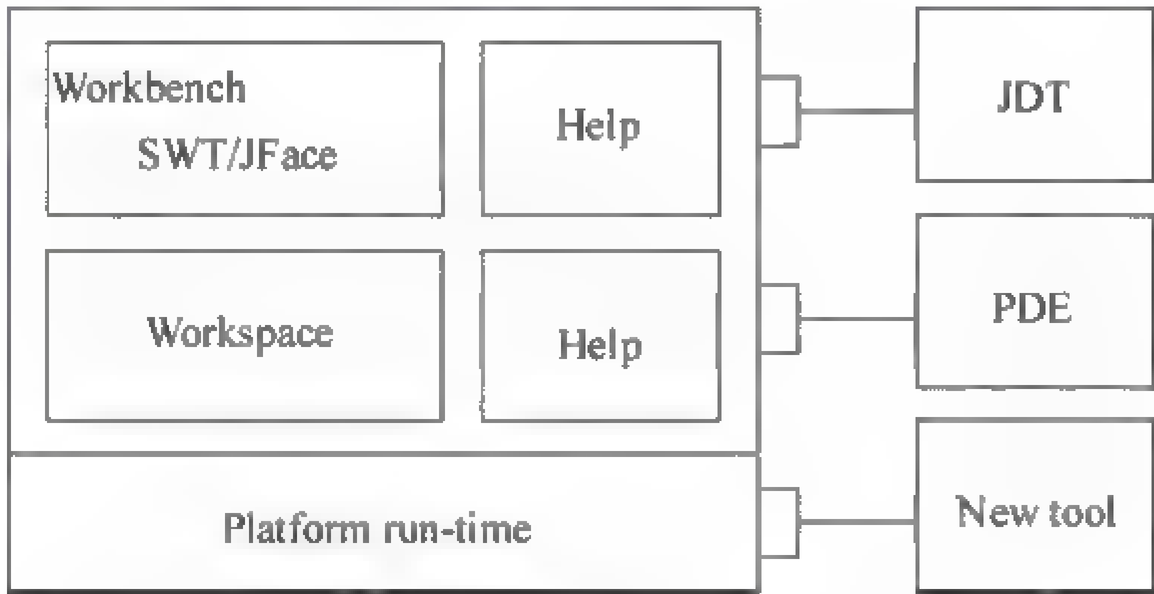

图5-5 Eclipse平台结构

在图5-5中：

（1）平台：平台运行库是内核，它在启动时检查已安装了哪些插件，并创建关于它们的注册表信息。同时，为了降低启动时间和资源使用，它在实际需要任何插件时才加载该插件。除了内核外，其他每样东西都是作为插件来实现的。

（2）工作区：工作区是负责管理用户资源的插件。这包括用户创建的项目，以及哪些项目中的文件、文件变更和其他资源。工作区还负责通知其他插件关于资源变更的信息，比如文件创建、删除或更改。

（3）工作台：工作台为Eclipse提供用户界面。它是使用标准窗口工具包（SWT）和一个更高级的API（JFace）来构建的，SWT是Java的Swing/AWT GUI API的非标准替代者，JFace则建立在SWT基础上，提供用户界面组件。

（4）SWT：已被证明是Eclipse最具争议的部分。SWT比Swing或SWT更紧密地映射到底层操作系统的本机图形功能，这不仅使得SWT更快速，而且使得Java 程序具有更像本

机应用程序的外观和感觉。但使用这个新的GUI API能会限制Eclipse工作台的可移植性，不过针对大多数流行操作系统的SWT移植版本已经可用。

（5）Eclipse：对SWT的使用只会影响Eclipse自身的可移植性——使用Eclipse构建的任何Java应用程序都不会受到影响，除非它们使用SWT而不是使用Swing/AWT。

（6）团队支持：团队支持组件负责提供版本控制和配置管理支持。它根据需要添加视图，以允许用户与所使用的任何版本控制系统（如果有的话）交互。大多数插件都不需要与团队支持组件交互，除非它们提供版本控制服务。

（7）帮助：帮助组件具有与Eclipse平台本身相当的可扩展能力。与插件向Eclipse添加功能相同，帮助提供一个附加的导航结构，允许工具以HTML文件的形式添加文档。

1．背景

Eclipse最初是由IBM公司开发的替代商业软件Visual Age for Java的下一代IDE开发环境，2001年11月贡献给开源社区，现在它由非营利软件供应商联盟Eclipse基金会（Eclipse Foundation）管理。2003年，Eclipse 3.0选择OSGi服务平台规范为运行时架构。 2007年6月，稳定版3.3发布；2008年6月发布代号为Ganymede的3.4版；2009年6月发布代号为Galileo的3.5版；2010年6月发布代号为Helios的3.6版。

Eclipse的基础是富客户机平台（Rich Client Platform，RCP）。RCP包括下列组件：核心平台（启动Eclipse，运行插件）、SGi（标准集束框架）、SWT（可移植构件工具包）、JFace（文件缓冲，文本处理，文本编辑器）、Eclipse工作台[即Workbench，包含视图（views）、编辑器（editors）、视角（perspectives）、和向导（wizards）]。

Eclipse采用的技术是IBM公司开发的（SWT），这是一种基于Java的窗口组件，类似Java本身提供的AWT和Swing窗口组件；不过IBM声称SWT比其他Java窗口组件更有效率。Eclipse的用户界面还使用了GUI中间层JFace，从而简化了基于SWT的应用程序的构建。

Eclipse的插件机制是轻型软件组件化架构。在富客户机平台上，Eclipse使用插件来提供所有的附加功能，例如支持Java以外的其他语言。已有的分离的插件已经能够支持C/C++（CDT）、PHP、Perl、Ruby，Python、telnet和数据库开发。插件架构能够支持将任意的扩展加入到现有环境中，例如配置管理，而绝不仅仅限于支持各种编程语言。

Eclipse的设计思想是一切皆插件。Eclipse核心很小，其他所有功能都以插件的形式附加于Eclipse核心之上。Eclipse基本内核包括：图形API（SWT/Jface），Java开发环境插件（JDT），插件开发环境（PDE）等。

2．Eclipse组件与计划

Eclipse通常由如下各种不同的计划组成[①]：

（1）Eclipse计划。本身包括Eclipse平台，Eclipse富客户端平台（RCP）和Java开发工具（JDT）。

（2）Eclipse测试和性能工具平台（TPTP）。提供一个允许软件开发者构建诸如测试调试、概况分析、基准评测等测试和性能工具的平台。

---

① http://www.eclipse.org/projects/listofprojects.php

（3）Eclipse 商业智能和报表工具计划（BIRT）。提供 Web 应用程序（特别是基于 Java 企业版的）的报表开发工具。

（4）Eclipse Web 工具平台计划（WTP）。用 Java 企业版 Web 应用程序开发工具来扩展 Eclipse 平台。它由以下部分组成：HTML、JavaScript、CSS、JSP、SQL、XML、DTD、XSD 和 WSDL 的源代码编辑器；XSD 和 WSDL 的图形界面编辑器；Java 企业版的"项目性质"（project nature）、建构器（builder）和模型（model），与一个 Java 企业版的导航（navigator）；一个 Web 服务（Web service）向导和浏览器，还有一个 WS-I 测试工具；最后是数据库访问查询的工具与模型。

（5）Eclipse 可视化界面编辑器计划（VEP）。一个 Eclipse 下创建图形用户界面代码生成器的框架。

（6）Eclipse 建模框架（EMF）。依据使用 XMI 描述的建模规格，生成结构化数据模型的工具和其他应用程序的代码。

（7）图形化编辑器框架（GEF）。能让开发者采用一个现成的应用程序模型来轻松地创建富图形化编辑器。

（8）UML 2。Eclipse 平台下的一个 UML 2.0 元模型的实现，用以支持建模工具的开发。

（9）AspectJ。一种针对 Java 的面向方面语言扩展。

（10）Eclipse 通讯框架（ECF）。专注于在 Eclipse 平台上创建通讯应用程序的工作、Eclipse 数据工具平台计划（DTP）和 Eclipse 设备驱动软件开发计划（DSDP）

（11）C/C++开发工具计划（CDT）。努力为 Eclipse 平台提供一个全功能 C 和 C++的集成开发环境（IDE），它使用 GCC 作为编译器。

（12）PHP 开发工具计划（PDT）。努力为 Eclipse 平台提供一个全功能 PHP 的集成开发环境（IDE）。

（13）Eclipse 平台 COBOL 集成开发环境子计划（COBOL）。将构建一个 Eclipse 平台上的全功能 COBOL 集成开发环境。

（14）并行工具平台（PTP）。将开发一个对并行计算机架构下的一组工具进行集成的平行工具平台，而且这个平台是可移植的，可伸缩的并基于标准的。

（15）嵌入式富客户端平台（eRCP）。计划将 Eclipse 富客户端平台扩展到嵌入式设备上。这个平台主要是一个富客户端平台（RCP）组件子集的集合。它能让桌面环境下的应用程序模型能够大致同样地能运用在嵌入式设备上。

3. Eclipse 是开源软件

开放源代码计划（Open Software Initiative）是一家非营利机构，它明确定义了开放源代码的含义及满足其标准的认证许可证（详见第 1 章）。Eclipse 是在 OSI 认可的通用公共许可证（CPL）1.0 版之下被授予许可的。它为 Eclipse 创建插件或将 Eclipse 用作为软件开发应用程序基础的开发人员，需要发布他们在 CPL 下使用或修改的任何 Eclipse 代码，但是可以自由决定自己添加的代码的许可证授予方式。与出自 Eclipse 的软件一起打包的专有代码不需要作为开放源代码来授予许可证，该源代码也不需要提供给用户。

通常情况下，开发人员不会使用 Eclipse 来开发插件，或创建基于 Eclipse 的新产品，但是 Eclipse 的开放源代码性质所意味的，并不只是它使得 Eclipse 免费可用。开放源代码鼓励创新，并激励开发人员为公共开放源代码库贡献代码。对此存在许多原因，不过最本质的原因或许是为这个项目作贡献的开发人员越多，这个项目就会变得对每个人都越宝贵。随着这个项目变得更加有用，更多的开发人员将会使用它，并围绕它形成一个社区，就像那些围绕 Apache 和 Linux 形成的社区一样。不过，现在以 Eclipse 为平台已经形成了 MyEclipse 商业集成软件。即 MyEclipse 企业级工作平台（MyEclipse Enterprise Workbench，简称 MyEclipse）是对 Eclipse IDE 的扩展，利用它可以在数据库和 JavaEE 的开发、发布，以及应用程序服务器的整合方面极大的提高工作效率；并且它是功能丰富的 JavaEE 集成开发环境，包括了完备的编码、调试、测试和发布功能，完整支持 HTML、Struts、JSP、CSS、Javascript、SQL、Hibernate 等。

Eclipse 的安装流程和方法可参见 http://public.dhe.ibm.com/software/dw/demos/Instal-lingEclipseEUP/InstallingEclipseEUP.pdf。

5.3.2 CVS

CVS（Concurrent Version System）诞生于 1986 年，当时作为一组 shell 脚本而出现，但它现在已经发展成了最流行的针对软件开发人员的源代码版本管理解决方案和实际的版本管理系统。CVS 是用于代码版本管理的开放源码的客户机/服务器解决方案，它可用于各种平台，包括 Linux 和 Windows NT/2000/XP。Eclipse 拥有与 Eclipse 平台 IDE 紧密集成的内置 CVS 客户机，它是作为一个单独透视图（CVS Repository Exploring 透视图）而实现的，用于与 CVS 的交互。目前，很多开源或者自由软件项目都使用 CVS 作为其程序员之间的中心点，以便能够综合各程序员的改进和更改。这些项目包括：Gnome、KDE、GIMP、Wine 等。CVS 的使用获 GNU 通用公共许可证授权。它是一个将一组文件放在层次目录树中以保持同步的系统。程序员可以从 CVS 服务器上更新他们的本地层次树副本，并将修改的结果或新文件发回；或者删除旧文件。

CVS 工作的基本思路在一台服务器上建立一个源代码库，库里可以存放许多不同项目的源程序。由源代码库管理员统一管理这些源程序。每个用户在使用源代码库之前，首先要把源代码库里的项目文件下载到本地，然后用户就可以在本地任意修改，最后用 CVS 命令进行提交，由 CVS 源代码库统一管理修改。这样，就好像只有一个人在修改文件一样，既避免了冲突，又可以做到跟踪文件变化等。

同样，与 CVS 相比，Subversion [①]也是一种开放源码的全新版本控制系统，支持可在本地访问或通过网络访问的数据库和文件系统存储库。Subversion 不但提供了常见的比较、修补、标记、提交、回复和分支功能性，还增加了追踪移动和删除的能力。此外，它支持非 ASCII 文本和二进制数据，所有这一切都使 Subversion 不仅对传统的编程任务非常有用，同时也适于 Web 开发、图书创作和其他在传统方式下未采纳版本控制功能的领域。

① http://subversion.tigris.org/

5.3.3 NetBeans

NetBeans[①]是一个始于 1997 年的 Xelfi 计划，本身是捷克布拉格查理大学 Charles University 的数学及物理学院的学生计划。此计划延伸而成立了一家公司进而发展这个商用版本的 NetBeans IDE，直到 1999 年 Sun 公司买下此公司。于是 NetBeans 由 Sun 公司在 2000 年创立，它是开放源运动以及开发人员和客户社区的家园，旨在构建世界级的 Java IDE。NetBeans 当前可以在 Solaris、Windows、Linux 和 Macintosh OS X 平台上进行开发，并在 SPL(Sun 公用许可)范围内使用。NetBeans IDE 目前支持 PHP、Ruby、JavaScript、Ajax、Groovy、Grails 和 C /C ++等开发语言。NetBeans 目前最新版本是 V 7.0.1。

在 NetBeans Platform 平台中，应用软件是用一系列的软件模块（modular software components）建构出来。而这些模块是一个 jar 档（Java archive file），它包含了一组 Java 程序的类，它们也实现了全依据 NetBeans 定义了的公开接口，以及一系列用来区分不同模块的定义描述档（manifest file）。这有利于模块化带来的好处，用模块来建构的应用程序，只要加上新的模块就能进一步扩充。由于模块可以独立地进行开发，所以由 NetBeans 平台开发出来的应用程序就能利用着第三方软件，非常容易及有效率地进行扩充。

NetBeans 平台是一种可重复使用的框架，用于简化其他桌面应用程序的开发。当基于 NetBeans 平台的应用被运行，平台主要类的 main 方法便会被运行。可用的模块会被放置在存储器中，并且开始运行任务。通常模块会只在被需要时，其代码才会被装进内存。并且整个 Netbeans 平台提供对桌面应用程序常用的服务，允许开发者集中于仅限于他的应用程序的逻辑设计。其中 NetBeans 平台的主要特征是：用户界面管理、用户设置管理、存储管理、视窗管理、向导框架。

5.3.4 Apache Ant

Apache Ant[②]是一个将软件编译、测试、部署等步骤联系在一起加以自动化的一个工具，大多用于 Java 环境中的软件开发。由 Apache 软件基金会所提供。默认情况下，XML 文件称为 build.xml。人们常把 Ant 与 Make 进行比较，Make 长期以来一直用于帮助自动完成构建过程。经过不同版本的改进，Ant 已发展成一个丰富的功能库，使其成为适用于许多场合的合适工具。例如，Ant 的当前版本（V1.6.2）提供的一些任务包括了操作文件内容、执行命令行和 Java 程序以及启动 SSH 和 FTP 连接的功能。

由于定义所有构建逻辑的 Ant 构建文件都是用 XML 编写的，因此，如果需要更改逻辑，则不需要重新编译代码，也不需要了解语言特定的语法。此外，Ant 具有高度的可扩展性。它提供了使用 Java 来创建自定义任务的功能，之后又可以通过与使用任何其他 Ant 任务相同的方式使用 Java。所有这些联系在一起意味着 Ant 是可以执行很多任务的极好选择。如程序清单 5-18 所示是使用 Ant 实现版权信息检查的部分脚本。

① http://netbeans.org/
② http://ant.apache.org/

程序清单 5-18：

```
<target name="checkLicense" >
  <for list="${scanFolderList}" param="folderList">
    <sequential>
      <for list="${scanFileType}" param="fileType">
        <sequential>
          <for param="file">
            <path>
              <fileset dir="@{folderList}" includes=**/*.@{fileType}>
                <not>
                  <contains text="${licenseFragment}" />
                </not>
              </fileset>
            </path>
        <sequential>
              <echo file="${reportFile}" message="@{file},${line.separator}"
                  append="true" encoding="UTF-8"/>
            </sequential>
          </for>
        </sequential>
      </for>
    </sequential>
  </for>
</target>
```

5.3.5　JUnit

JUnit[1]是由 Erich Gamma 和 Kent Beck 编写的一个开源的单元测试框架，是逐渐成为源于 Kent Beck 的 sUnit 的 xUnit 家族中为最成功的一个。它属于白盒测试，只要将待测类继承 TestCase 类，就可以利用 JUnit 的一系列机制进行便捷的自动测试了。主要用于测试期望结果的断言（Assertion）、用于共享共同测试数据的测试工具、用于方便的组织和运行测试的测试套件、图形和文本的测试运行器。并且 JUnit 是在极限编程和重构（refactor）中被极力推荐使用的工具，因为在实现自动单元测试的情况下可以大大地提高开发的效率，但是实际上编写测试代码也是需要耗费很多的时间和精力的，图 5-6 所示是 JUnit 的结构。

目前，JUnit 在测试开发领域的核心地位日渐稳定。不仅 Eclipse 将 JUnit 作为默认的 IDE 集成组件，而且基于 JUnit 的各种测试框架也在业内被广泛应用，并获得了一致好评。同时，JUnit 的设计精简，易学易用，但是功能却非常强大，这归因于它内部完善的代码结构。Erich Gamma 是著名的 GOF（Gang of Four）之一，因此 JUnit 中深深渗透了扩展性优良的设计模式思想。JUnit 提供的 API 既可以写出测试结果明确的可重用单元测试用例，也提供了单元测试用例成批运行的功能。在已经实现的框架中，用户可以选择三种方式来显示测试结果，

① http://www.junit.org/

并且显示的方式本身也是可扩展的[1]。

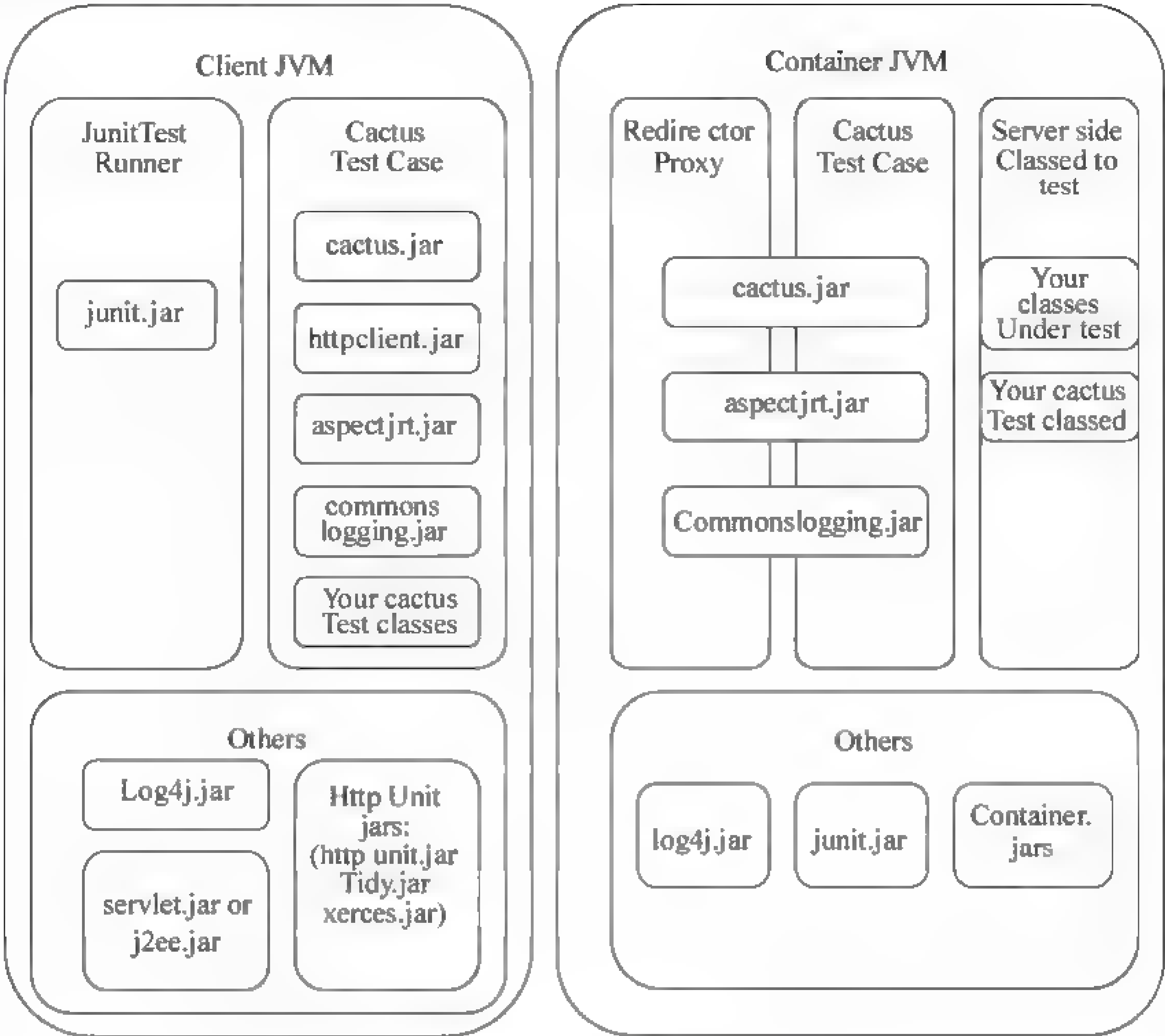

图 5-6　JUnit 结构

5.4　常用的支持服务器软件

前面概述、总结了开源软件的语言、开发环境，现进一步概述、结构开源软件所支持的服务器软件，因为对许多开源软件来讲，是需要在服务器上运行，才能构成跨平台、多异构间的通信。特别是基于 Web 有应用程序必须由服务器来跑，才能让面向开源软件的程序真正运起来。下面介绍当前中小型机构常用的几种服务器软件。

5.4.1　Tomcat

Tomcat[2]是大家非常熟悉的一款稳定性好、性能优越之一的 Web 服务器开源软件。它是由 Apache 软件基金会下属的 Jakarta 项目开发的一个 Servlet 容器，按照 Sun Microsystems 提供的技术规范，实现了对 Servlet 和 JavaServer Page（JSP）的支持，并提供了作为 Web 服务器的一些特有功能，如 Tomcat 管理和控制平台、安全域管理和 Tomcat 阀等。由于 Tomcat 本身也内含了一个 HTTP 服务器，它也可以被视作一个单独的 Web 服务器。但是，不能将

① http://www.ibm.com/developerworks/cn/java/j-lo-junit-src/

② http://tomcat.apache.org/

Tomcat 和 Apache Web 服务器混淆，Apache Web Server 是一个用 C 语言实现的 HTTP web server；这两个 HTTP web server 不是捆绑在一起的。Apache Tomcat 包含了一个配置管理工具，也可以通过编辑 XML 格式的配置文件来进行配置。同时，最新的 Servlet 和 JSP 规范总是能在 Tomcat 中得到体现，Tomcat 5 支持最新的 Servlet 2.4 和 JSP 2.0 规范。因为 Tomcat 技术先进、性能稳定，而且免费，因而深受 Java 爱好者的喜爱并得到了部分软件开发商的认可，成为目前比较流行的 Web 应用服务器。目前最新版本是 V7.0

5.4.1.1　Tomcat 原理概述

Tomcat 被 JavaWorld 杂志的编辑选为 2001 年度最具创新的 java 产品（Most Innovative Java Product），同时它又是 Sun 公司官方推荐的 servlet 和 jsp 容器（具体可以见 http://java.sun.com/products/jsp/tomcat/）。Tomcat 的结构很复杂，但是 Tomcat 也非常的模块化，图 5-7 就是 Tomcat 结构图。在图中看出 Tomcat 是由 Connector 和 Container 两个组件构成。Connector 组件是可以被替换的，这样可以提供给服务器设计者更多的选择。因为这个组件是重要的，不仅跟服务器的设计本身有关，而且和不同的应用场景也十分相关，所以一个 Container 可以选择对应多个 Connector。多个 Connector 和一个 Container 就形成了一个 Service，有了 Service 就可以对外提供服务了，但是 Service 还要一个生存的环境，必须要有 Server 能够给它运行基础。所以整个 Tomcat 的生命周期由 Server 控制。而 Service 接口主要是为了关联 Connector 和 Container，同时会初始化它下面的其他组件，而接口中它并没有规定一定要控制它下面的组件的生命周期，所有组件的生命周期在一个 Lifecycle 的接口中控制。Tomcat 中 Service 接口的标准实现类是 StandardService，它不仅实现了 Service 借口，同时还实现了 Lifecycle 接口，这样它就可以控制它下面的组件的生命周期了。

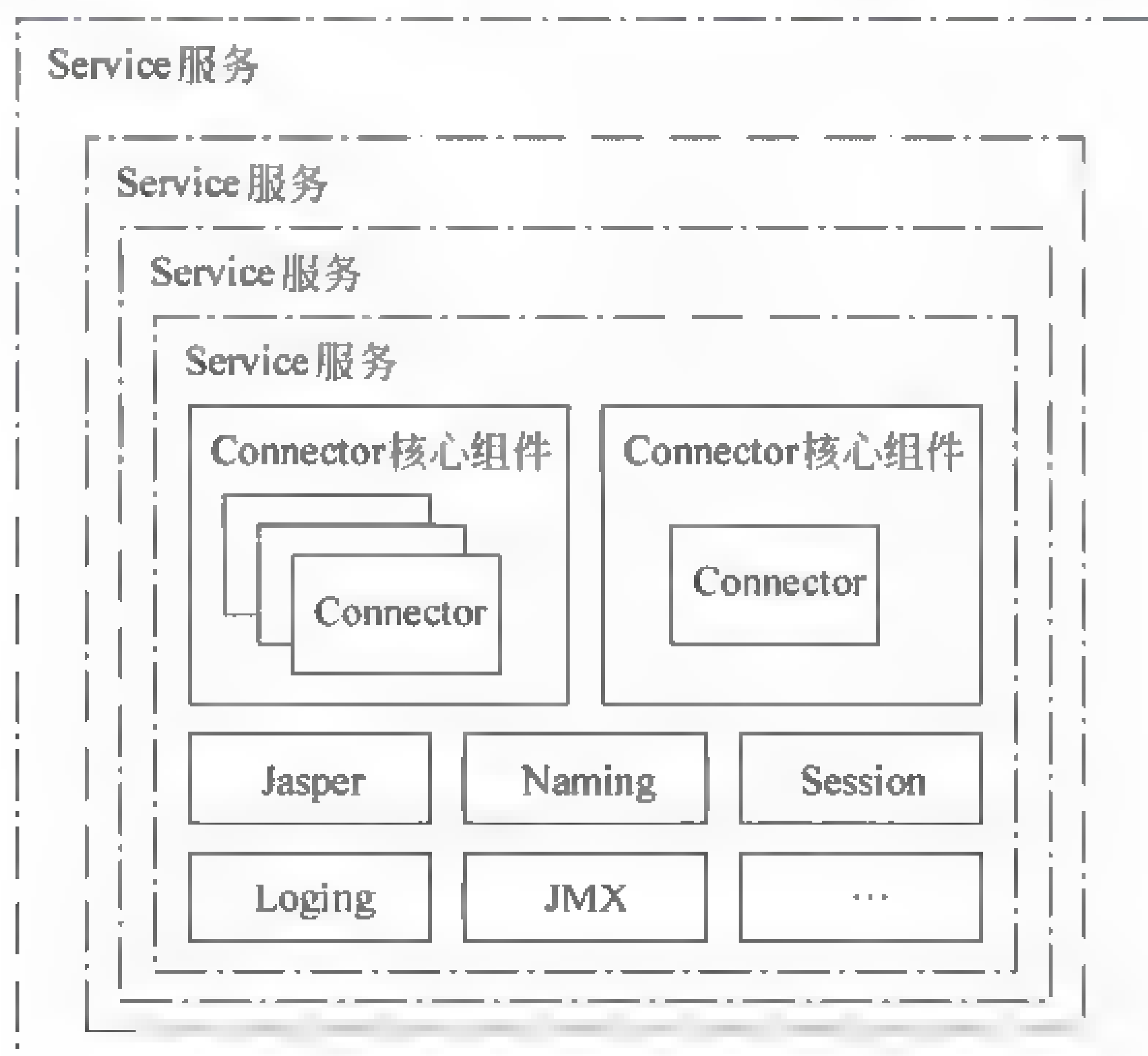

图 5-7　Tomcat 结构

1. Connector 组件

Connector 组件是 Tomcat 中两个核心组件之一，它的主要任务是负责接收浏览器的发过来的 tcp 连接请求，然后创建一个 Request 和 Response 对象分别用于和请求端交换数据，然后会产生一个线程来处理这个请求，并把产生的 Request 和 Response 对象传给处理这个请求

的线程，处理这个请求的线程就是 Container 组件要完成的功能。图 5-8 是 Connector 处理一次请求顺序图。

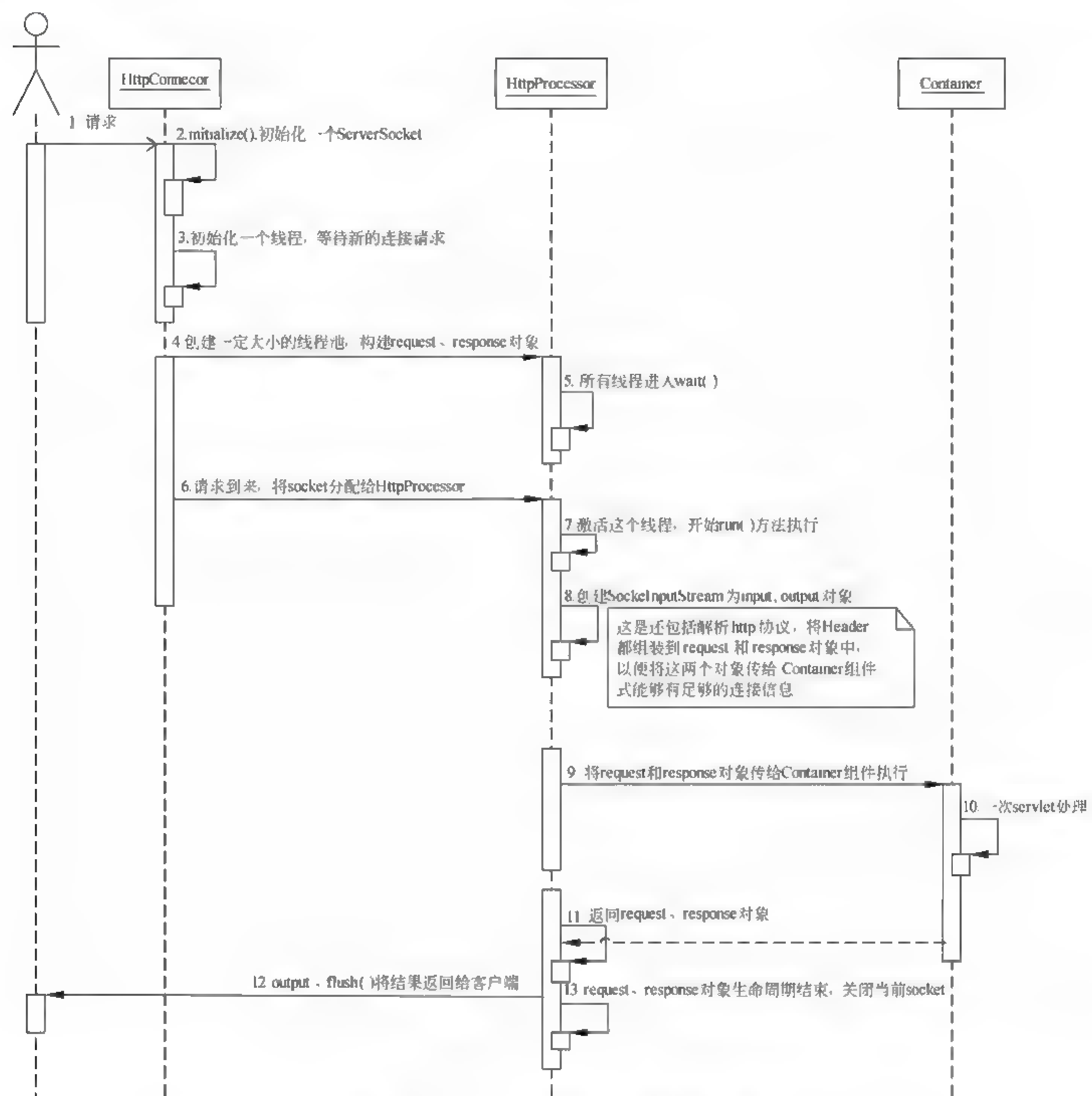

图 5-8　Connector 处理一次请求顺序图

Tomcat5 以后的版本中默认的 Connector 是 Coyote，这个 Connector 是可以选择替换的。Connector 最重要的功能就是接收连接请求然后分配线程让 Container 来处理这个请求，所以这必然是多线程的，多线程的处理是 Connector 设计的核心。Tomcat 将这个过程更加细化，它将 Connector 划分成 Connector、Processor、Protocol，另外 Coyote 也定义自己的 Request 和 Response 对象。

Container 是容器的父接口，所有子容器都必须实现这个接口，Container 容器的设计用的是典型的责任链的设计模式，它有四个子容器组件构成，图 5-9 所示是这四个容器的关系，分别是：Engine、Host、Context、Wrapper，这四个组件不是平行的，而是父子关系，Engine 包含 Host，Host 包含 Context，Context 包含 Wrapper。通常一个 Servlet class 对应一个 Wrapper，如果有多个 Servlet 就可以定义多个 Wrapper，如果有多个 Wrapper 就要定义一个更高的

Container。

Context 还可以定义在父容器 Host 中，Host 不是必须的，但是要运行 war 程序，就必须要 Host，因为 war 中必有 web.xml 文件，这个文件的解析就需要 Host 了，如果要有多个 Host 就要定义一个 top 容器 Engine 了。而 Engine 没有父容器了，一个 Engine 代表一个完整的 Servlet 引擎。当 Connector 接受到一个连接请求时，将请求交给 Container，图 5-10 是 Container 的这个过程的时序图。

图中各接口（斜体字表接口）与类（正体字表类）的属性与方法
请参见：http://tomcat.apache.org/tomcat-6.0-doc/index.html

图 5-9　四个容器的关系图

图 5-10　Engine 和 Host 处理请求的时序图

在图 5-10 中，StandardEngineValve 和 StandardHostValve 是 Engine 和 Host 的默认的 Valve，它们是最后一个 Valve 负责将请求传给它们的子容器，以继续往下执行，图 5-11 是看 Context 和 Wrapper 容器时如何处理请求的时序图。

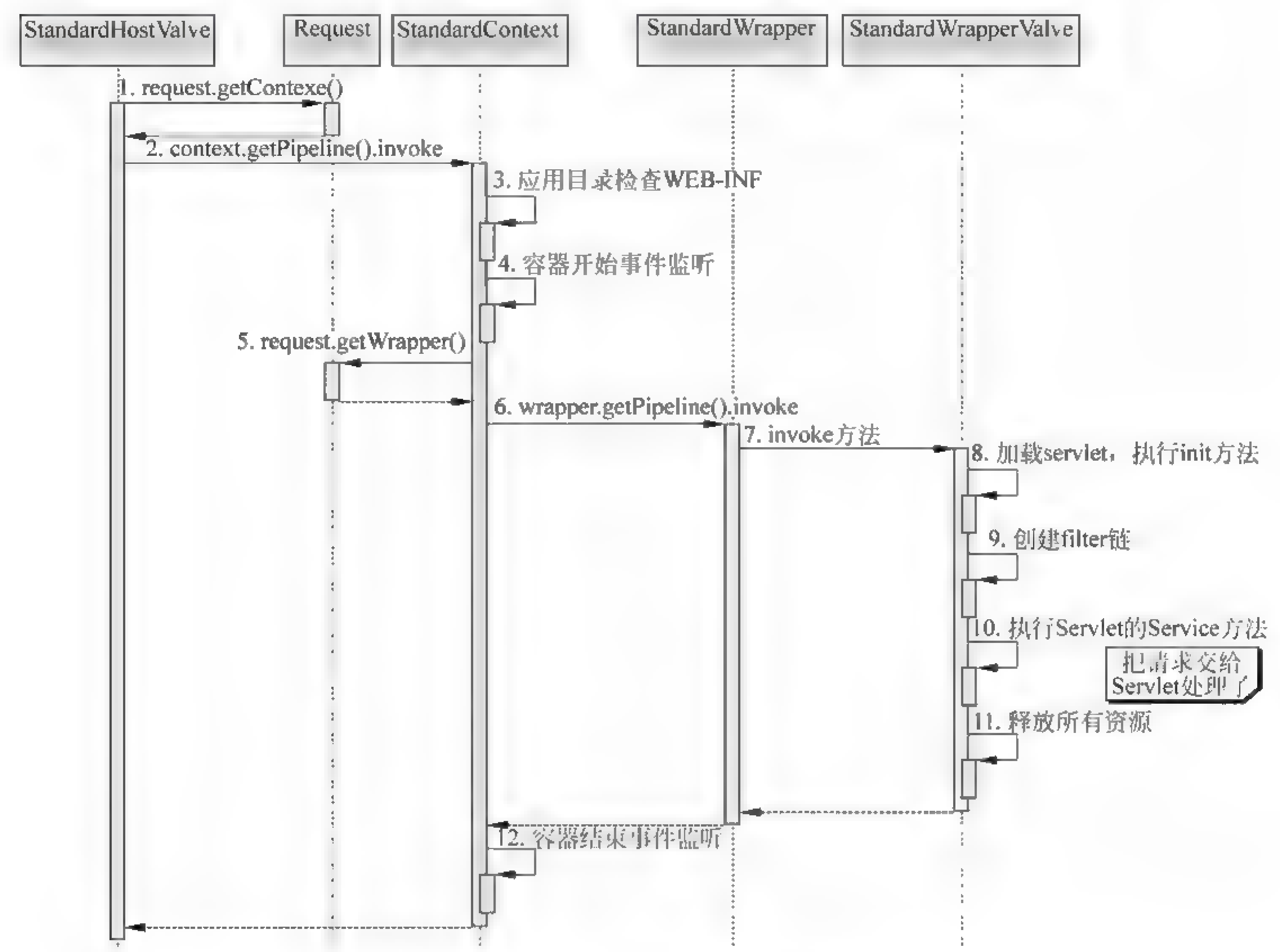

图 5-11 Context 和 wrapper 的处理请求时序图

2. Tomcat 的容器

Tomcat 的容器主要包括 Engine、Host、Context、Wrapper 这四个容器。同时，Tomcat 还有其他重要的组件，如安全组件 security、logger 日志组件、session、mbeans、naming 等其他组件，这些组件共同为 Connector、Container 和容器提供必要的服务。

（1）Engine 容器。Engine 容器比较简单，它的标准实现类是 StandardEngine，这个类 Engine 没有父容器了，如果调用 setParent 方法时将会报错。图 5-12 所示是 Engine 容器接口的类型。

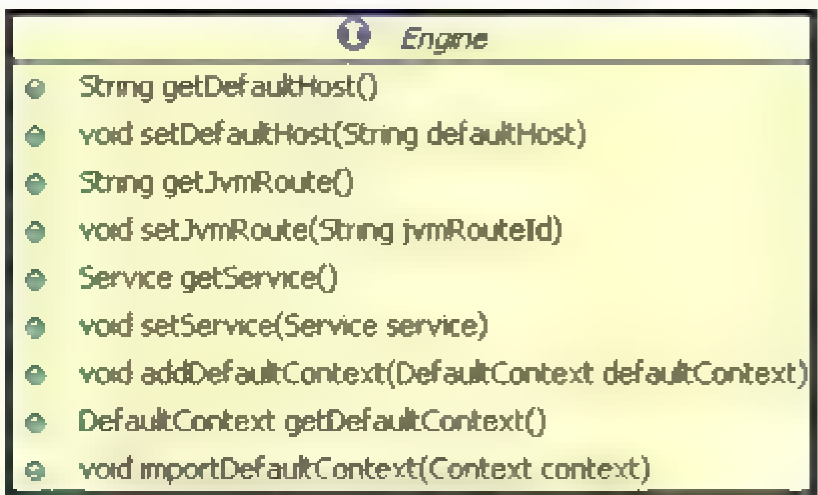

图 5-12 Engine 容器接口

（2）Host 容器。Host 是 Engine 的子容器，一个 Host 在 Engine 中代表一个虚拟主机，这个虚拟主机的作用就是运行多个应用，它负责安装和展开这些应用，并且标识这个应用以便能够区分它们。它的子容器通常是 Context，它除了关联子容器外，还有就是保存一个主机应该有的信息，图 5-13 是 Host 容器关系图。

图 5-13　Host 相关的类图

在图 5-14 中可以看出除了所有容器都继承的 ContainerBase 外，StandardHost 还实现了 Deployer 接口，上图清楚的列出了这个接口的主要方法，这些方法都是安装、展开、启动和结束每个 web application。并且 Deployer 接口的实现是 StandardHostDeployer，这个类实现了的最重要的几个方法，Host 可以调用这些方法完成应用的部署等。

（3）Context 容器。Context 代表 Servlet 的 Context，它具备了 Servlet 运行的基本环境，理论上只要有 Context 就能运行 Servlet 了。简单的 Tomcat 可以没有 Engine 和 Host。Context 最重要的功能就是管理它里面的 Servlet 实例，Servlet 实例在 Context 中是以 Wrapper 出现的，还有一点就是 Context 如何才能找到正确的 Servlet 来执行它呢？Tomcat5 以前是通过一个 Mapper 类来管理的，Tomcat5 以后这个功能被移到了 request 中，在前面的时序图中就可以发现获取子容器都是通过 request 来分配的。

（4）Wrapper 容器。Wrapper 代表一个 Servlet，它负责管理一个 Servlet，还包括的 Servlet 的装载、初始化、执行和资源回收。Wrapper 是最底层的容器，它没有子容器了，所以调用它的 addChild 将会报错。Wrapper 的实现类是 StandardWrapper，它还实现了拥有一个 Servlet

初始化信息的 ServletConfig，由此看出 StandardWrapper 将直接和 Servlet 的各种信息打交道。图 5-14 所示是 ServletConfig 与 StandardWrapperFacade、StandardWrapper 的关系。

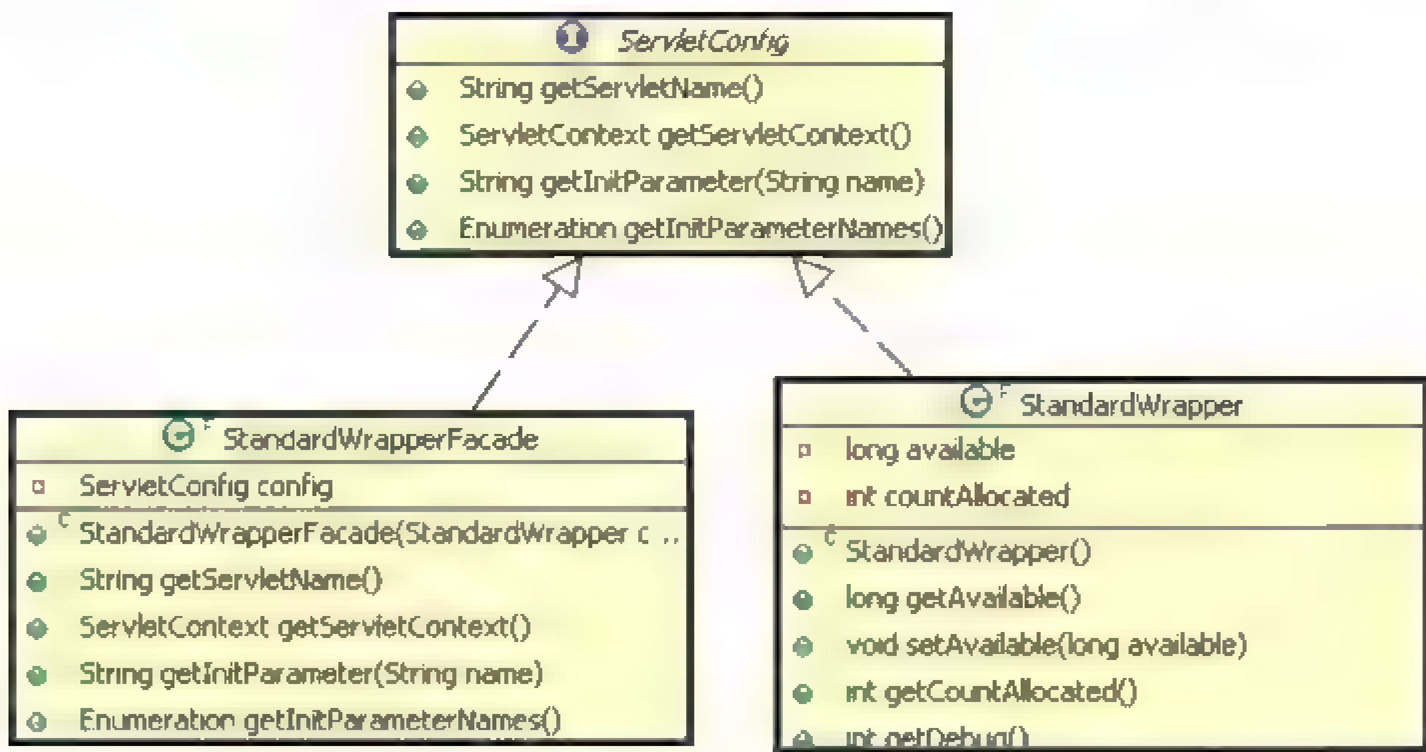

图 5-14　ServletConfig 与 StandardWrapperFacade、StandardWrapper 的关系

Servlet 可以获得的信息都在 StandardWrapperFacade 中封装，这些信息又是在 StandardWrapper 对象中获得的。所以 Servlet 可以通过 ServletConfig 获得有限的容器的信息。并且当 Servlet 被初始化完成后，就等着 StandardWrapperValve 去调用它的 service 方法了，当调用 service 方法之前要调用 Servlet 所有的 filter。

5.4.1.2　Tomcat 结构

前面概述了 Tomcat 的基本原理，下面进一步概述 Tomcat 的文件结构和 Server.xml 释义，表 5-2 所示是 Server.xml 主要属性解释。

表 5-2　Server.xml 的标签主要属性解释

元素名	说明	属性	解释
server		port	指定一个端口，这个端口负责监听关闭 tomcat 的请求
		shutdown	指定向端口发送的命令字符串
service		name	指定 service 的名字
Connector	表示客户端和 service 之间的连接	port	指定服务器端要创建的端口号，并在这个断口监听来自客户端的请求
		minProcessors	服务器启动时创建的处理请求的线程数
		maxProcessors	最大可以创建的处理请求的线程数
		enableLookups	如果为 true，则可以通过调用 request.getRemoteHost() 进行 DNS 查询来得到远程客户端的实际主机名，若为 false 则不进行 DNS 查询，而是返回其 ip 地址
		redirectPort	指定服务器正在处理 http 请求时收到了一个 SSL 传输请求后重定向的端口号
		acceptCount	指定当所有可以使用的处理请求的线程数都被使用时，可以放到处理队列中的请求数，超过这个数的请求将不予处理
		Connection-Timeout	指定超时的时间数（以 ms 为单位）

续表

元素名	说明	属性	解释
Engine	表示指定service中的请求处理机，接收和处理来自Connector的请求	defaultHost	指定默认的处理请求的主机名，它至少与其中的一个host元素的name属性值是一样的
	Context表示一个web应用程序，通常为WAR文件，关于WAR的具体信息见servlet规范	docBase	应用程序的路径或者是WAR文件存放的路径
		path	表示此web应用程序的url的前缀，这样请求的url为http://localhost:8080/...
		reloadable	这个属性非常重要，如果为true，则tomcat会自动检测应用程序的/WEB-INF/lib和/WEB-INF/classes目录的变化，自动装载新的应用程序，我们可以在不重启tomcat的情况下改变应用程序
host	表示一个虚拟主机	name	指定主机名
		appBase	应用程序基本目录，即存放应用程序的目录
		unpackWARs	如果为true，则tomcat会自动将WAR文件解压，否则不解压，直接从WAR文件中运行应用程序
Logge	表示日志，调试和错误信息	className	指定logger使用的类名，此类必须实现org.apache.catalina.Logger接口
		prefix	指定log文件的前缀
		suffix	指定log文件的后缀
		timestamp	如果为true，则log文件名中要加入时间
Realm	表示存放用户名，密码及role的数据库	className	指定Realm使用的类名，此类必须实现org.apache.catalina.Realm接口
Valve	功能与Logger差不多，其prefix和suffix属性解释和Logger中的一样	className	指定Valve使用的类名，如用org.apache.catalina.valves.AccessLogValve类可以记录应用程序的访问信息
		directory	指定log文件存放的位置
		pattern	有两个值，common方式记录远程主机名或ip地址，用户名，日期，第一行请求的字符串，HTTP响应代码，发送的字节数。combined方式比common方式记录的值更多

5.4.2　Geronimo

Apache Geronimo[①]是Apache软件基金会的J2EE服务器开放源代码，它集成了众多先进技术和设计理念，当然也可以集成Tomcat。这些技术和理念大多源自独立的项目，使得配置和部署模型也各不相同。但Geronimo能将这些项目和方法的配置及部署且完全整合到一个统一、易用的模型中。同时，作为符合J2EE标准的服务器，Geronimo提供了丰富的功能集和无责任Apache许可，具备立即部署式的J2EE 1.4容器的各种优点，其中包括：符合J2EE1.4

① http://geronimo.apache.org/

标准的服务器、预集成的开放源码项目、统一的集成模型、可伸缩性、可管理性和配置管理功能等。目前，Apache Geronimo 作为一个开放源码解决方案正在迅速发展，新的 2.2.1 版本已经完成了，使得 Geronimo 已经度过了原始时期。然而像 Geronimo 这样的大型开放源码解决方案总是受到大量开发人员的关注，使得开发人员无论是进行提交，还是为了内部使用或业务使用而进行开发，他们都需要更多地了解 Geronimo 的结构，从而掌握构建过程，图 5-15 是 Geronimo 结构图。

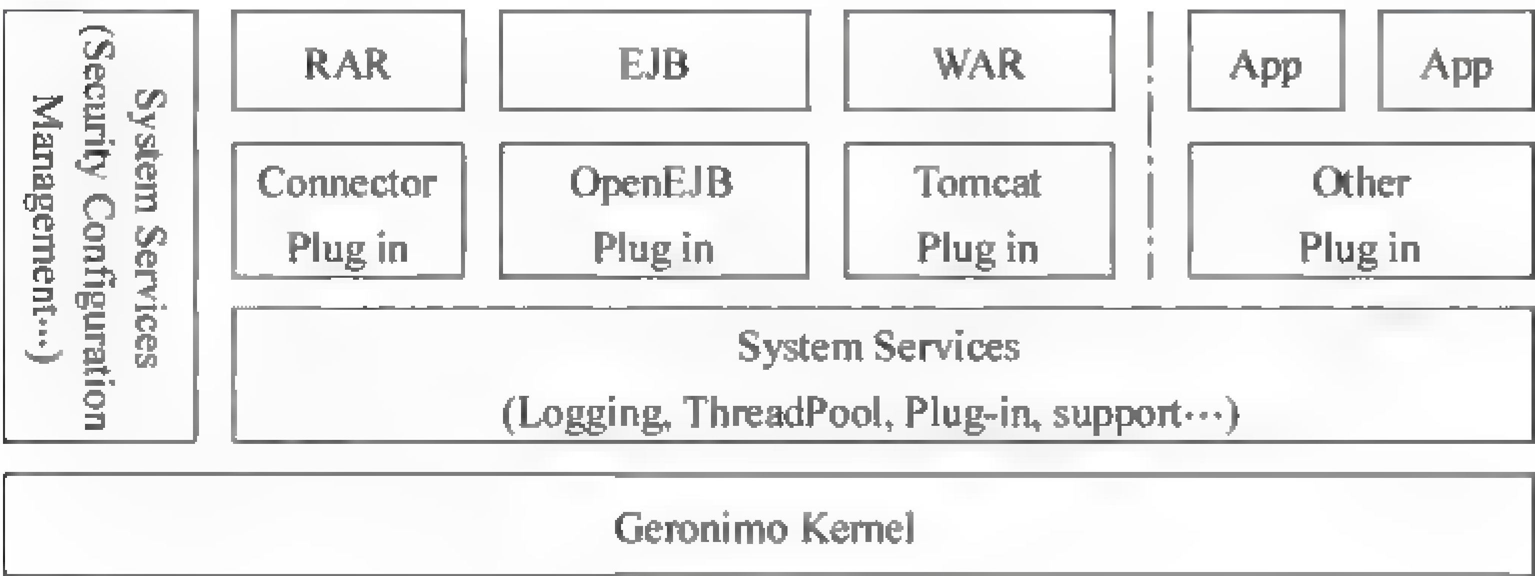

图 5-15　Geronimo 结构

Geronimo 包括二进制和源代码两部分，二进制下载包含 Geronimo 的核心，这是运行和使用 Geronimo 所必须的；源代码下载包含所有 Geronimo 源代码，以及包括用于构建整个树的 Maven 构建脚本。其中：

二进制代码：二进制发行版的文件/目录比较少，但是比较大，因为它们是二进制形式的。下面列出相关目录：

（1）bin——可执行文件。

（2）config-store —— 存储 Geronimo 的部署者，包括应用程序的 .war 和其他文件。

（3）doc —— GBeans 的一些计划或配置放在这里。

（4）repository —— 这里包含构建和运行 Geronimo 所需的依赖项。

（5）var —— 这里存储配置和属性文件，以及内置的 Derby 数据库系统。

源代码发行版包括：

（1）applications —— 容纳 Geronimo 附带的各种标准应用程序。

（2）modules —— 包含几个目录，其中包含所有 Geronimo 源代码。

（3）Assembly ——主构建目录，也可以单独构建每个目录。

表 5-3 所示是 Geronimo 中集成的开放源码项目。如程序清单 5-19 是 Geronimo 连接 JDBC 的程序片段。在程序中的 XML 文件中有一个 dependency 元素，该元素指定了 JDBC 驱动程序.jar 所在的库目录的 URI 路径。在具体应用时要对它进行修改，使它满足所需要的的数据库，即编辑 UserName、Password、Driver 以及 ConnectionURL 的 configId 和 config-property-setting 元素。

表 5-3　Geronimo 集成的开源项目和集成在 Geronimo 的项目

开放源码项目	说明
Apache Tomcat	支持 Java Servlet 2.4 和 JSP2.0 的 Web 层应用服务器
Jetty	支持 Java Servlet 2.4 和 JSP 2.0 的 Web 层应用服务器——可以替代 Tomcat 服务器
ActiveMQ	开放源码的 Java Message Service（JMS）1.1 应用程序提供者，支持消息驱动 bean（MDB）

续表

开放源码项目	说明
OpenEJB	开放源码的 Enterprise JavaBeans（EJB）容器系统和 EJB 服务器，支持 Enterprise JavaBeans 2.1，包括 Container Managed Persistence 2（CMP2）和 EJB Query Language（EJBQL）
Apache Axis、Scout	一种 Simple Object Access Protocol（SOAP）实现（Axis）和 JSR 93（JAXR）实现（Scout），用于对 Web 服务和 Web Services Interoperability Organization（WS-I）Basic Profile 支持
Apache Derby	完全成熟的关系数据库管理系统（RDBMS），支持本机 Java Database Connectivity（JDBC）
Spring Framework	流行的应用程序框架，用于从轻量级 Inversion of Control（IoC）组件构建应用程序
ServiceMix	开放源码工具集，支持 Java Business Integration（JBI）并且为面向服务体系结构（SOA）实现提供 Enterprise Service Bus（ESB）

程序清单 5-19：

```
<?xml version="1.0">
<connector xmlns="http://geronimo.apache.org/xml/ns/j2ee/connector"
  version="1.5"
  configId="MysqlDatabase"
  parentId="org/apache/geronimo/Server">
<dependency>
  <uri>
    mysql/jars/mysql-connector-java-3.1.8-bin.jar
  </uri>
</dependency>
<resourceadapter>
  <outbound-resourceadapter>
    <connection-definition>
      <connectionfactory-interface>
        javax.sql.DataSource
      </connectionfactory-interface>
      <connectiondefinition-instance>
        <name>MysqlDataSource</name>
        <config-property-setting name="UserName">
          geronimo
        </config-property-setting>
        <config-property-setting name="Password">
          geronimo
        </config-property-setting>
        <config-property-setting name="Driver">
          com.mysql.jdbc.Driver
        </config-property-setting>
        <config-property-setting name="ConnectionURL">
          jdbc:mysql://localhost/geronimo
        </config-property-setting>
        <config-property-setting name="CommitBeforeAutocommit">
```

```
            false
          </config property setting>
          <config property-setting name="ExceptionSorterClass">
            org.tranql.connector.NoExceptionsAreFatalSorter
          </config-property-setting>
          <connectionmanager>
            <local-transaction/>
            <single-pool>
              <max-size>8</max-size>
              <min-size>1</min-size>
              <blocking-timeout-milliseconds> 1000
               </blocking-timeout-milliseconds>
               <idle-timeout-minutes>
                 30
               </idle-timeout-minutes>
               <match-one/>
            </single-pool>
          </connectionmanager>
          <global-jndi-name>
            jdbc/MysqlDatabase
          </global-jndi-name>
        </connectiondefinition-instance>
      </connection-definition>
    </outbound-resourceadapter>
  </resourceadapter>
  </connector>
```

5.4.3 Jboss

JBoss[1]是一套应用程序服务器，属于开源的企业级 Java 中间件软件，用于实现基于 SOA 架构的 web 应用和服务，也包含一组可独立运行的软件。2006 年 4 月 10 日，Redhat 宣布斥资 3.5 亿美元收购 JBoss。同时，JBoss 是全世界开发者共同努力的成果，一个基于 J2EE 的开放源代码的应用服务器，并得到了 J2EE 的认证服务器，因为 JBoss 代码遵循 LGPL 许可，可以在任何商业应用中免费使用它，而不用支付费用。Jboss 也是一个管理 EJB 的容器和服务器，支持 EJB 1.1、EJB 2.0 和 EJB 3.0 的规范。但 JBoss 核心服务不包括支持 servlet/JSP 的 WEB 容器，在这种情况下，一般与 Tomcat 或 Jetty 绑定使用。

5.4.4 Jetty

Jetty[2]这个项目成立于1995年，现在已经有非常多的成功产品是基于Jetty的，比如Apache

① http://www.jboss.org/

② http://jetty.codehaus.org/jetty/

Geromino，JBoss，IBM Tivoli，Cisco SESM 等。Jetty 可以用来作为一个传统的 Web 服务器，也可以作为一个动态的内容服务器，并且 Jetty 可以非常容易的嵌入到 Java 应用程序当中。而且 Jetty 是一个开源的 servlet 容器，它为基于 Java 的 web 内容提供服务支持，例如为 JSP 和 servlet 提供运行环境。Jetty 是使用 Java 语言编写的，它的 API 以一组 JAR 包的形式发布。并且具有丰富功能的 Http 服务器和 Web 容器，也可以免费的用于商业行为。开发人员可以将 Jetty 容器实例化成一个对象，并可以迅速为一些独立运行（stand-alone）的 Java 应用提供网络和 Web 连接。下面概述 Jetty 的特性和 Jetty Continuations 机制原理。

1. Jetty 特性

Jetty 特性主要包括易用性、可扩展性和易插入性，其中：

（1）易用性。Jetty 设计的基本原则主要体现在通过 XML 或者 API 来对 Jetty 进行配置，通常默认配置可以满足大部分的需求，使得将 Jetty 嵌入到应用程序当中时，只需要非常少的代码就可以实现。

（2）可扩展。在使用 Ajax 的 Web 2.0 的应用程序中，每个连接需要保持更长的时间，这样线程和内存的消耗量会急剧的增加。这就使得我们担心整个程序会因为单个组件陷入瓶颈而影响整个程序的性能。但是有了 Jetty，即使在有大量服务请求的情况下，系统的性能也能保持在一个可以接受的状态。并且可以利用 Continuation 机制来处理大量的用户请求和时间比较长的连接。另外，Jetty 设计了非常良好的接口，因此在 Jetty 的某种实现无法满足用户的需要时，用户可以非常方便地对 Jetty 的某些实现进行修改，使得 Jetty 适用于特殊的应用程序的需求。

（3）易嵌入性。Jetty 设计之初就是作为一个优秀的组件来设计的，这也就意味着 Jetty 可以非常容易的嵌入到应用程序当中，而不需要程序为了使用 Jetty 做来修改。从某种程度上，程序员可以把 Jetty 理解为一个嵌入式的 Web 服务器，程序清单 5-20 是 Jetty 的程序片段。

程序清单 5-20：

```
public class JettyServer {
    public static void main(String[] args) {
        Server server = new Server(8080);
        server.setHandler(new DefaultHandler());
        XmlConfiguration configuration = null;
        try {
            configuration = new XmlConfiguration(
                new FileInputStream("C:/development/Jetty/jetty-6.1.6rc0/etc/
                jetty.xml"));
        } catch (FileNotFoundException e1) {
            e1.printStackTrace();
        } catch (SAXException e1) {
            e1.printStackTrace();
        } catch (IOException e1) {
            e1.printStackTrace();
        }

        try {
```

```
            configuration.configure(server);
            server.start();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

在程序片段中，创建一个 Server 对象，并设置端口为 8080，然后为这个 Server 对象添加一个默认的 Handler。接着我们用配置文件 jetty.xml 对这个 server 进行设置，最后我们使用方法 server.start()将 Server 启动起来就可以了。从这段代码可以看出，Jetty 是非常适合用于作为一个组件来嵌入到我们的应用程序当中的，这也是 Jetty 的一个非常重要的特点。

2．Jetty Continuations 机制原理

Jetty 的 Continuation 机制，首先需要提到 Ajax 技术，Ajax 技术是当前开发 Web 应用的非常热门的技术，也是 Web 2.0 的一个重要的组成部分。Ajax 技术中的一个核心对象是 XMLHttpRequest 对象，这个对象支持异步请求，所谓异步请求即是指当客户端发送一个请求到服务器的时候，客户端不必一直等待服务器的响应。这样就不会造成整个页面的刷新，给用户带来更好的体验。

Jetty 利用 Java 语言的非堵塞 I/O 技术来处理并发的大量连接。Jetty 有一个处理长连接的机制：一个称为 Continuations 的特性。利用 Continuation 机制，Jetty 可以使得一个线程能够用来同时处理多个从客户端发送过来的异步请求。但要使用 Continuations，必须对 Jetty 进行配置，以使用其 SelectChannelConnector 处理请求。这个连接器构建在 java.nio API 之上，因此使它能够不用消耗每个连接的线程就可以持有开放的连接。当使用 SelectChannelConnector 时，ContinuationSupport.getContinuation()将提供一个 SelectChannelConnector.RetryContinuation 实例。当对 RetryContinuation 调用 suspend()时，它将抛出一个特殊的运行时异常 RetryRequest，该异常将传播到 servlet 以外，并通过过滤器链传回，并由 SelectChannelConnector 捕获。但是发生该异常之后并没有将响应发送给客户机，请求被放到处于等待状态的 Continuation 队列中，而 HTTP 连接仍然保持打开状态。此时，为该请求提供服务的线程将返回 ThreadPool，用以为其他请求提供服务，程序清单 5-21 是 Continuation 机制。

程序清单 5-21：

```
public class ChatContinuation extends HttpServlet{
    public void doPost(HttpServletRequest request, HttpServletResponse
    response){
        postMessage(request, response);
    }
    private void postMessage(HttpServletRequest request, HttpServletResponse
    response)
    {
        HttpSession session = request.getSession(true);
        People people = (People)session.getAttribute(session.getId());
        if (!people.hasEvent())
```

```
            {
                Continuation continuation =
                    ContinuationSupport.getContinuation(request, this);
                people.setContinuation(continuation);
                continuation.suspend(1000);
            }
            people.setContinuation(null);
            people.sendEvent(response);
        }
    }
```

5.4.5　Derby

Apache Derby[①]是 Apache 软件基金会所研发的开放源码数据库管理系统；由于 Derby 是一个纯 Java 程式，因此只需要操作系统支援 Java 虚拟机，Derby 便可执行。Derby 是特别地为 Java 环境进行优化，Derby 本身不仅是一个纯 Java 程式，而且 Derby 在执行用户的 SQL 程式时，能够把 SQL 编译成 Java bytecode 并以系统的 Java 虚拟机执行。由于 SQL 程式转成的 Java bytecode 能被 JIT 动态翻译，因此 Derby 可能比传统的数据库管理系统更佳的性能。其功能主要包括支援主从架构或嵌入环境，多线程，ACID（原子性、一致性、隔离性和持久性），Java 数据库连接（JDBC），低系统需求。

因为 Apache Derby 是用 Java 语言编写的，所以可以在任何存在合适的 Java 虚拟机（JVM）的地方运行。这意味着 Derby 实际上可以在任何操作系统上运行，包括 Microsoft Windows、Macintosh、Linux 和 UNIX 平台。Derby 也可以在三个 Java 平台的任何一个上运行：Java 2 Platform, Micro Edition（J2ME）、Java 2 Platform, Standard Edition（J2SE）和 J2EE。Derby 软件绑定在 Java 档案（JAR）文件中，只有 2 MB 大小。由于内存占用小，所以 Derby 数据库可以容易地与应用程序绑定在一起。程序清单 5-22 是初始化 Apache Derby 的代码，图 5-16 所示是 Apache Derby 结构。

程序清单 5-22：

```
public class SampleAction implements IWorkbenchWindowActionDelegate {
private int queryRecords()
        throws SQLException, IllegalAccessException, ClassNotFoundException,
        InstantiationException {
    Connection currentConnection = null;
    System.setProperty("derby.system.home",
        Sample derbyPlugin.getDefault().getStateLocation().toFile().
        getAbsolutePath());
    Properties props = new Properties();
    try {
        Class.forName(DRIVER).newInstance();
        currentConnection = DriverManager.getConnection(PROTOCOL
```

① http://db.apache.org/derby/

```
                + DATABASE, props);
        } catch (SQLException sqlException) {
            //trying to create database
            currentConnection = DriverManager.getConnection(PROTOCOL
                + DATABASE + ";create=true", props);
            try {
                Statement s = currentConnection.createStatement();
                try {
                    s.execute(CREATE TABLE);
                } finally {
                    s.close();
                }
                currentConnection.commit();
            } catch (SQLException ex) {
                currentConnection.close();
                throw ex;
            }
        }
        return 0;
    }
```

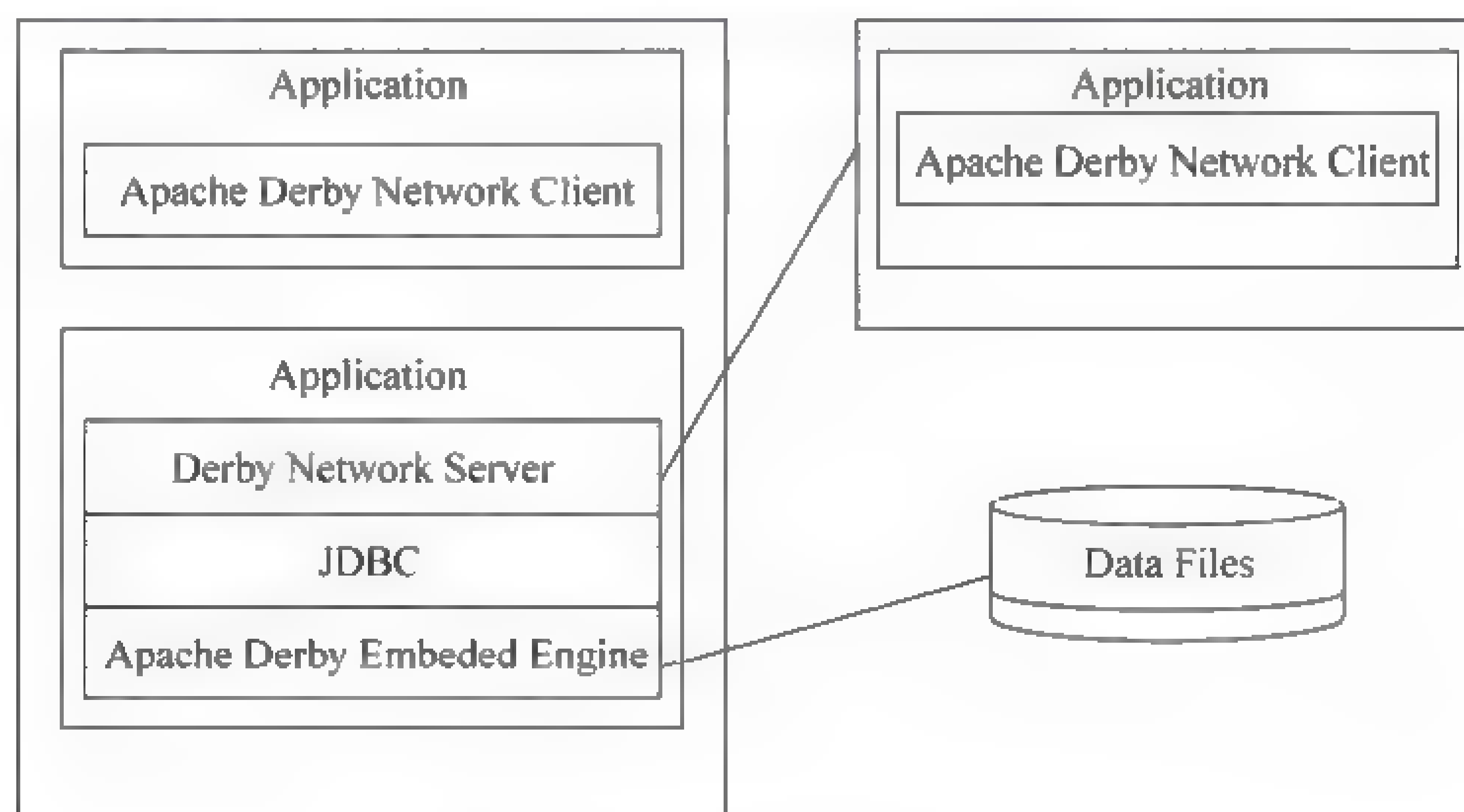

图 5-16　Apache Derby 结构

5.5　Web 2.0 技术

Web 2.0 最初由 O'Reilly 公司在 2003 年提出，在 2004 年召开 Web 2.0 大会之后，这个词逐渐流行了起来。Web 2.0 事实上是指基于 Web 的下一代社区和托管服务，诸如社会网络、维基百科、大众分类等。它包括一系列原则、模式、创新和实践，代表着新一代的以 Web 为基础的技术，简化并促进了 Web 用户参与分享、交互与协作，从而产生更有价值的内容和服务。2.0 暗示着 Web 的升级换代，也指软件开发人员和最终用户在使用互联网平台的方式上发生了巨大的变化。简单的说，Web 2.0 就是更为简单地对数据源的访问、使用和协作。最

终在 2005 年 O'Reilly 和 Battelle 总结了他们认为表现了 Web 2.0 应用特色的一些关键原则：

（1）将以 Web 作为平台；

（2）驾驭群体智能；

（3）数据将变成未来的“Intel Inside”；

（4）软件不断发行与升级的循环将会终结；

（5）轻量型程序设计模型；

（6）通过内容和服务的联合使轻量的业务模型可行；

（7）软件运行将跨越单一设备；

（8）丰富的用户体验；

（9）分享和参与的架构所驱动的网络效应；

（10）通过带动分散的、独立的开发者把各个系统和网站组合形成大汇集的改革；

（11）快速的反应与新增功能的便捷。

Web 2.0 的应用可以让人了解目前万维网正在进行的一种改变，即从一系列网站到一个成熟的为最终用户提供网络应用的服务平台。这种概念的支持者期望 Web 2.0 服务将在很多用途上最终取代桌面计算机应用。Web 2.0 并不是一个技术标准，不过它包含了技术架构及应用软件。它的特点是鼓励信息最终利用者通过分享，使得可供分享的资源变得更丰盛。Web 2.0 也是网络运用的新时代，也使网络成为了新的应用平台，内容是因为每位用户的参与（Participation）而产生，参与所产生的个人化（Personalization）内容，借由人与人（P2P）的分享，就形成了现在 Web 2.0 的世界[1]。

目前，Web 3.0 也正在发展和形成中。Web 3.0 一词包含多层含义，用来概括互联网发展过程中可能出现的各种不同的方向和特征，包括将互联网本身转化为一个泛型数据库，以及跨浏览器、超浏览器的内容投递和请求机制，人工智能技术的运用，语义网，地理映射网，运用 3D 技术搭建的网站，甚至虚拟世界或网络公国等。

5.5.1　Web 2.0 实现的相关技术

Web 2.0 就在我们身边，当浏览并写作博客、在亚马逊网上书店阅读图书评论、在 Wiki 上更新一个词条或者从网站订阅 RSS 的时候，事实上已经在使用 Web 2.0 技术了。Web 2.0 已经融入到了我们的生活、学习、工作和娱乐之中了。其中的国内：豆瓣网、百度百科、人人网、新浪微博、优酷网、猪八戒威客网；和国外：维基百科、Twitter、Facebook、Google 阅读器、flickr、del.icio.us、Skype、Amazon.com Web Services。下面概述总结 Web 2.0 的特征和技术。

1．Web 2.0 特征

Web 2.0 和 Web 1.0 之间的不同，可以概括出 Web 2.0 成功的八个核心原则：

（1）群众智慧（Collective Intelligence）：建立参与架构，借助网络效应和算法，使得软件的用户越多，而服务和质量也变得越好。

（2）数据，下一个“Intel Inside”：利用独特、难以复制的数据源，使数据变得跟功能一

---

① http://radar.oreilly.com/archives/2006/12/web-20-compact.html

样重要，成为核心竞争能力。

（3）“复合”创新，建立平台：通过数据和服务的重新组合，创造新的市场和机会。

（4）丰富用户体验：超越传统的Web界面模式，让在线应用拥有与桌面应用一样的丰富用户体验。

（5）支持多种设备：支持各种连接到互联网的设备，为用户提供无所不在、无缝的在线体验。

（6）软件即服务（Software as a Services，SaaS）和永久试验版（Perpetual Beta）：改变了传统软件开发和使用的模式，转向永久在线、持续更新和软件即服务的模式。

（7）利用长尾：借助互联网带来的接触极大规模客户的能力以及极低成本的营销方式，来获得细分的niche市场的利润。

（8）轻量级模型和低成本优势的可扩充能力：利用轻量级的商业模型和软件开发模式来快速、廉价地构造产品和服务。

而且以上模式可以由下列几个Web 2.0的特质相互关联起来：

（1）大规模互连：网络效应使得边际同核心一样重要，颠覆着旧的通信、发布、分发和聚合模式。

（2）去中心化：大规模互联颠覆着传统的控制和权力结构，带来更大程度的去中心化。使得系统更多地从通过边沿的拉动来生长，而不是借助核心的推动向外生长。

（3）以用户为中心：网络效应给予用户前所未有的力量，他们参与、对话、协作，最终产生巨大的影响。

（4）开放：这种开放性，是以因特网的开放技术标准为基础的，但很快地演进到一个由开放应用所构成的生态系统，这些应用建构在开放数据、开放 API 和可重用的组件之上。

（5）轻量级：软件由小团队使用敏捷方法设计和开发，使用简单数据格式和协议，采用运行开销小的平台和框架，应用和服务部署简易，商业上力图保持低的投资和成本，营销上利用简单的消费者之间的口口相传来形成病毒式传播。

（6）自然浮现：不是依靠预先完整定义好的应用结构，而是让应用的结构和行为随着用户的实际使用而灵活适应和自然演变；成功来自合作，而不是控制。

2. Web 2.0技术

Web 2.0技术主要包括：博客（BLOG）、RSS、wiki百科全书（Wiki）、网摘、社会网络（SNS）、P2P、即时信息（IM）、基于地理信息服务（LBS）等。

Web 2.0主要特点和基于这些特点产生的具有代表性的服务：

（1）博客（Blog）：最早期的Web 2.0服务之一，可使任何参与者拥有自己的专栏、成为网络内容产生源，进而形成微媒体，为网络提供文字、图片、声音或视频信息。

（2）内容源（RSS）：伴随博客产生的简单文本协议，将博客产生的内容进行重新格式化输出，从而将内容从页面中分离出来，便于同步到第三方网站或提供给订阅者进行阅读。

（3）Wiki：是一个众人协作的平台，方便写百科全书、词典等。

（4）参与评论与评分（Digg）：Digg、分享机制和去中心化是 Web 2.0 最显著地特点，Digg机制为更多网络用户提供了参与网络建设的机制，无须进行内容贡献或创作，只要用户对网络内容进行评分或点评，即可参与到网络内容的建设过程当中。

（5）网摘（Delicious）：用户可根据自己的喜好进行网络内容的收藏或转载，并将自己的

收藏或转载整理成列表、分享给更多的用户，从而在网络上起到信息聚合与过滤的作用。

（6）社会化网络（SNS）：以原有的网站、内容为中心，转而侧重于以人与人之间的关联为中心，网络上每一个结点所承载的不再是信息，而是以具体的自然人为结点，形成的新型互联网形态。

（7）微博（Twitter）：博客的精简版，有较为严格的字数限制。更简单的发布流程和更随意的写作方式，使得参与到网络内容贡献中的门槛降低，更大程度的推动了网络内容建设和个体信息贡献。

（8）地理信息签到服务（LBS）：集地理信息系统（GIS）、微博（Twitter）和移动设备（Mobile）以及 A-GPS 定位服务于一身的增强型微博系统，其主导思想是每一条信息不仅利用时间为索引，同时也加入了地理经纬度的索引，从而不仅可通过时间对信息进行筛选、亦可利用地理坐标对信息进行筛选。

（9）支持 Web 服务技术：双向的消息协议是 Web 2.0 架构的关键元素之一，两个主要的类型是 REST 和 SOAP 方法。REST（Representational State Transfer）表示了一种 Web 服务客户端传送所有的事务的状态。SOAP（Simple Object Access Protocol）和类似的轻量方法都依赖服务器来保存状态信息。在这两种情况下，服务是通过一个应用程序接口（API）来调用的。这个 API 常常是根据网站的特殊需求定义的，但是标准的 Web 服务 API 的 API 依然被广泛使用。一般来说 Web 服务的通用语言是 XML，但并不一定，这是因为还存在大量不同的其他语言，如 JSON，YAML 等。

5.5.2　Web 2.0 用户界面定制工具

在用户界面方面，当今的企业应用程序开发人员受到来自用户和运营部门的双重压力。一方面，代表用户的业务部门希望应用程序具有丰富的用户界面，能够最大限度地提高用户的工作效率。他们希望所有应用程序都表现得像 Microsoft 的 Excel 或者其他客户机应用程序一样，并希望应用程序能够提供即时响应。此外，若有相同数据的多个视图（例如，一个表格视图和一个图形视图），那么还希望在其中一个视图中进行修改时，其他视图能够立即反映出这一修改。另一方面，IT 运营部门喜欢纯粹的基于服务器的交付模型。尽管他们知道 HTML 用户体验不如基于本机操作系统（OS）的用户界面那么健壮，但他们认为为了改进用户体验，安装、配置和管理客户机代码的成本太高了。这时 Web 正好能很好的来有效解决这两个问题，下面就介绍几种用户界面定制技术。

1. Flex 和 OpenLaszlo

前面章节已经概述总结了 Flex，现将作为一种界面定制技术进行讲解。不过，Flex 和 OpenLaszlo 是极其相似的声明式方法，用来为 Java EE 应用程序创建比浏览器更好的用户体验。Flex 由 Adobe/Macromedia 提供，而 OpenLaszlo 是最初由 Laszlo Systems Inc 创建的开放源码软件。在这两种环境中，都使用独特的基于 XML 的语法来布置和创建用户界面。为了允许不同的 UI 元素与服务器进行交互和通信，可以用 ActionScript（Flex）或 JavaScript（OpenLaszlo）编写脚本。

尽管这两种技术有许多相似性，但关键的一点差异是 Flex 和 OpenLaszlo 所需要的运行环境不同。对于需要与服务器交换数据的客户机，Flex 需要一个 Flex Data Services Server，

它与 Flash Player 插件中运行的客户机进行通信。在本质上，这个服务器为客户机和应用程序的服务器组件之间的所有通信和数据交换提供中介。而 OpenLaszlo 对版本 4 做了一些运行时的改进，使它对于开发人员更具吸引力。一项改进是版本 3 引入了一种 SOLO 开发模式，使得在某些部署配置中不再需要 Laszlo Presentation Server。另一个主要的改进是客户机运行时环境。从 OpenLazlo 4 后，它使基于 Laszlo 的应用程序能够不带 Adobe/Macromedia Flash Player 插件运行。

例如：添加一个按键的两种表示方式：

```
Flex：可以用 MXML（Multimedia XML）编写以下代码：<a name="code-text"><mx:Button
label="Submit"</mx:Button></a>。
OpenLaszlo：可以用 LZX（LasZlo XML）编写以下代码：<a name="code-text"><button
<Submit</ button></a>。
```

2．Faces Client Components

JavaServer Faces（JSF）是一种 Java EE 1.4 组件，最初是作为 JSR 127 开发的。这种技术的关键目标是降低 Java EE 应用程序开发用户界面时要求 Java 开发人员具备的技能水平。因为 JSF 是一个框架，它提供了许多开箱即用的功能；而在过去，开发人员在用 Java Server Pages（JSP）构建同样的用户界面时需要手工编写这些功能。

JSF 框架提供了一个 DataTable 部件，可以用来显示数据。并且当用户点击 Next 来显示表中的后 x 行数据时，JSF 框架将会处理 Next 请求，程序员不必自己编写任何代码。尽管 JSF 简化了创建丰富的 HTML 用户界面的过程，但是根据设计 JSF 是一种基于服务器的技术。对后 x 行数据的请求从浏览器发送到服务器，JSF 框架代码需要在服务器上处理这个请求，这时 JSF 需要一次到服务器的请求/响应往返。

3．Ajax

Ajax（Asynchronous JavaScript + XML：异步 JavaScript 和 XML）是 Jesse James Garrett 创造的一个术语，它是指一种基于标准的技术/设计模式，用来为服务器部署的应用程序开发比浏览器更好的用户体验。Ajax 对服务器技术没有什么要求，可以处理 Java EE 应用程序、.Net 应用程序和其他应用程序。通过使用 Ajax，可以编写 JavaScript（JS）代码来改进 HTML，从而创建出丰富的交互性用户体验。例如，JavaScript 可以执行本地用户输入验证，为相同的数据提供不同的视图（条形图、表格、饼图等等），或者通过浏览器的 XMLHTTP -Request 对象与应用程序的服务器组件进行异步的交互。后续章节也将进一步概述和总结 AJAX。

4．Ext JS

Ext JS 是一种强大的 JavaScript 库，它通过使用可重用的对象和部件简化了 Ajax 开发，并且 Ext JS 是一种 JavaScript 开发框架，通过这种强大的 JavaScript 库从而达到可重用的对象和部件简化 Ajax 开发。Ext JS 最初是 Jack Slocum 编写的一组 Yahoo! User Interface（YUI）Library 扩展。但是，随着 2.0 版的发布，它已经成为市场上最简单最强大的 JavaScript 库。

Ext JS 采用 Open Source LGPL 3.0 许可证的条款，如果打算在另一个开放源码项目或者个人、教育或非盈利项目中使用 Ext JS，这是最合适的许可。如果希望在项目中使用 Ext JS 时避免开发源码许可证的某些限制，或者由于内部原因必须拥有许可证，或者希望在商业上

支持 Ext JS 的开发，这是使用商用许可证。如果打算对 Ext JS 进行重新打包，或者作为软件开发库销售 Ext JS，这时需要始设备生产商（OEM）/转售商许可证。

例如，若 Ext JS 安装在 Web 服务器上的 lib/ext 目录中，则：

```
<script type ="text/javascript" src="lib/ext/ext-base.js"></script>
<script type ="text/javascript" src="lib/ext/ext-all.js"></script>
```

5. ECMAScript for XML（E4X）

E4X[1]是一种对 JavaScript 的简单扩展，这使得编写 XML 脚本非常的简单。它实际上就是在 JavaScript 中添加了对 XML 的直接支持。它同时也是一种 ECMA（European Computer Manufactures Association）标准。在编程环境中考虑处理 XML 时，可以简化 DOM（Document Object Model）的 path 等操作的复杂性。这时因为组合了许多最好的 DOM 特性和极其简单的数据绑定，为在浏览器中处理 XML 提供了一种更简便的方法，如程序清单 5-23 是使用 E4X 的程序片段。

程序清单 5-23：

```
<script type="text/javascript;e4x=1">
myquestion = <question>
        <display>Is it animal, vegetable, or mineral?</display>
        <answerOption>Animal</answerOption>
        <answerOption>Vegetable</answerOption>
        <answerOption>Mineral</answerOption>
    </question>;
alert("The question is '" + myquestion.display + "'");
</script>
```

5.5.3　Web 2.0 页面处理技术

JavaScript 是目前应用的最广的 Web 客户端开发技术之一。它能够给 Web 界面带来更多的动态效果，让 Web 应用变得更加易于使用和快速响应。随着 Ajax 技术的流行以及 Web 2.0 应用，开发人员越来越多地认识到 JavaScript 的强大功能。相应地，许多开发人员开始研究如何更好、更方便地使用 JavaScript。因此出现了许多强大的 JavaScript 开发工具包，如 Prototype、jQuery、Dojo、Ext JS 等。它们的主要目的是让 JavaScript 的开发变得更加简单、直观。通过对 JavaScript 基本功能的封装、集合、改造，这些开发开发工具包的 API 可以让开发人员只通过简单的几行代码就可以实现原本需要大量代码才能实现的功能。这种开发方式将极大地提高工作效率，也会吸引更多地开发人员来使用它们。下面进一步介绍 jQuery 和 Dojo 的 Web 2.0 页面处理技术。

1. jQuery

jQuery 由 John Resig 创建于 2006 年初，对于任何使用 JavaScript 代码的程序员来说，它是一个非常有用的 JavaScript 库。它有助于简化 JavaScript 以及 Ajax 编程。与类似的

① http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-357.pdf

JavaScript 库不同，jQuery 具有独特的基本原理，可以简洁地表示常见的复杂代码，并且希望获得一个能解决文档对象模型 DOM Ajax 开发中的一些复杂问题库。

jQuery 能帮助程序员保证代码简洁易读，再也不必编写大堆重复的循环代码和 DOM 脚本库调用。使用 jQuery，可以把握问题的要点，并使用尽可能少的代码来实现需要的功能。表 5-4 所示是一个单击（click）事件，用纯 JavaScript 代码、DOM 脚本来实现和 jQuery 比较。

表 5-4　用纯 JavaScript 代码、DOM 脚本来实现和 jQuery 比较

没有使用 jQuery 的 DOM 脚本	使用了 jQuery 的 DOM 脚本

var external_links =
document.getElementById('external_links');
var links =
external_links.getElementsByTagName('a');
for (var i=0;i < links.length;i++) {
var link = links.item(i);
link.onclick = function() {
return confirm('You are going to visit: ' +
this.href);
};}
	$('#external_links a').click(function() {
	return confirm('You are going to visit: ' + this.href);
	});

2. Dojo

Dojo（即 Unified toolkit）①是一个用 JavaScript 编写的开放源码 DHTML 工具包。可以使用 Dojo 在 Web 页面和支持 JavaScript 的任何其他环境中轻松地构建动态功能。通过使用 Dojo 提供的组件，可以提高 Web 站点的易用性、响应性和功能性。可以轻松地构建可分解的用户界面，并快速地构建交互式组件和转换。同时，可以使用 Dojo 提供的低级 API 和兼容性层编写可移植的 JavaScript，并简化复杂的脚本。Dojo 提供多个入口点和很有前景的 API，它们独立于解释器，而且非常关注消除使用方面的障碍。因此，Dojo 在基于 Web 的应用程序中越来越受到欢迎，同时 Dojo 还具有如下优点：

（1）Dojo 是基于 JavaScript 的工具包，能够满足对集成简便性的需求。只须包含 JavaScript，开发人员就可以享受到一个强大 API 带来的好处，这个 API 对于大多数开发任务应该足够了。它使团队能够开发出功能丰富、外观漂亮的组件，而且很容易把这些组件集成到应用程序中。

（2）Dojo 支持 Ajax，这意味着应用程序的响应性更好，总体效率更高。更重要的是可以与主应用程序非常快速地交互。Ajax 是一种用来创建交互式 Web 应用程序的 Web 开发技术。它在幕后与服务器交换少量数据，这样就不必在用户每次发出请求时都重新装载整个 Web 页面，从而使 Web 页面显得响应性更好。这种技术会增加 Web 页面的交互性、速度、功能和易用性。程序清单 5-24 是 Dojo 一程序片段的基本使用方法。

程序清单 5-24：

```
function addOnJsFiles(file)
{
```

① http://dojotoolkit.org

```
    var scriptTag = document.createElement('script');
    scriptTag.src  = file;
    scriptTag.type = 'text/javascript';
    scriptTag.defer = true;
    document.getElementsByTagName('head').item(0).appendChild(scriptTag);
}/*启用 dojo.js 的包含)*/
...
addOnJsFiles('js/dojo/dojo.js');
var email = /((([a-zA-Z0-9 \-])+\@(([a-zA-Z0-9\-])+\.)+([a-zA-Z0-9]{2,4})+)/g
/*搜索所有有效的电子邮件模式*/
...
var tmpDiv = document.getElementById(divid);
var FloatingPaneWidget = dojo.widget.createWidget("FloatingPane",
    {
        id:"panel",windowState:"minimized",
        title:"Send Email", hasShadow: "true",
        resizable:"true",displayMinimizeAction:"true",
        toggle:"explode",constrainToContainer: "false"    },
    tmpDiv);/*创建 Dojo 方法*/
...
```

5.5.4　RSS 技术

RSS[①]（Really Simple Syndication，简易信息聚合）是一种基于 XML 消息来源格式规范，用以发布经常更新数据的网站，例如博客文章、新闻、音频或视频的网摘。RSS 文件（又称摘要、网络摘要、或频更新，提供到频道）包含了全文或是节录的文字，再加上发行者所订阅的网摘数据和授权的元数据。网络摘要能够使发行者自动地发布他们的数据，同时也使读者能更够定期更新他们喜欢的网站或是聚合不同网站的网摘。RSS 摘要可以借由 RSS 阅读器、feed reader 或是 aggregator 等网页或以桌面为架构的软件来阅读。而标准的 XML 档式可允许信息在一次发布后通过不同的程序阅览，从而使用户借由将网摘输入到 RSS 阅读器或是用鼠标点取浏览器上指向订阅程序的 RSS 小图标之后，由 URI（非通常称为 URL）来订阅网摘。并使 RSS 阅读器定期检阅是否有更新，然后下载给监看用户界面，并最终通过投放广告为网站赢利。表 5-5 所示是几种常用见的阅读器。

表 5-5　常见 RSS 阅读器

名称	说明	网址
Google Reader	支持中文界面、持 HTTPS、阅读速度快	http://www.google.com/reader
Bloglines	RSS 或 Atom 取得内容、不同于客户端的新闻聚合软件各自取得内容，Bloglines 会统一在服务器端定时更新	于 2010 年 10 月关闭
NewsGator	英文界面，速度不怎么样，对中文支持不太好。阅读界面上会显示 Google Adsense 的广告	http://www.newsgator.com

① http://cyber.law.harvard.edu/rss/rss.html

续表

名称	说明	网址
Rojo	英文界面，速度慢。中文支持极其不好，阅读界面上会显示 Google Adsense 的广告	http://www.rojo.com
抓虾	一个国内的阅读器，浏览速度还可以	http://www.zhuaxia.com

RSS 是 Internet 上连锁内容和元数据的一种格式。通常用于共享标题和到新闻文章的链接。对于新闻文章，真正的文章不一定是共享的，但是关于文章的元数据通常是共享的；这种元数据可以包含标题、URL 或者摘要。对于出版商而言，RSS 是一种重要的工具，因为它可用于连锁内容，并把第三方的内容集成到系统中。同时，RSS 是一种 XML 语言。所有的 RSS 文件必须符合万维网联盟（W3C）Web 站点上发布的 XML 1.0 规范。

一个 RSS 文件就是一段规范的 XML 数据，该文件一般以 rss，xml 或者 rdf 作为后缀。RSS 是在线共享内容的一种简易方式（又称聚合内容，Really Simple Syndication），通常在时效性比较强的内容上使用 RSS 订阅能更快速获取信息。而网站提供的 RSS 输出信息，有利于让用户获取网站内容的最新更新，这样网络用户可以在客户端借助于支持 RSS 的新闻聚合工具软件（例如 SharpReader,NewzCrawler，FeedDemon），在不打开网站内容页面的情况下阅读支持 RSS 输出的网站内容。下面是编写 RSS 的样式：

RSS 文件由一个<channel>元素及其子元素组成。除了频道内容本身之外，<channel>还以项的形式包含表示频道元数据的元素，例如<title>、<link>、<description>。而项通常是频道的主要部分，包含经常变化的内容。

（1）频道：一般有三个元素，提供关于频道本身的信息：

① <title>：频道或提要的名称。

② <link>：与该频道关联的 Web 站点或者站点区域的 URL。

③ <description>：简要介绍该频道是做什么的。

许多频道子元素都是可选的。常用的<image>元素包含三个必需的子元素：

① <url>：表示该频道的 GIF、JPEG 或 PNG 图像的 URL。

② <title>：图像的描述。当频道以 HTML 呈现时，用作 HTML <image>标签的 ALT 属性。

③ <link>：站点的 URL。如果频道以 HTML 呈现，该图像作为到这个站点的链接。

<image>还有三个可选的子元素：

① <width>：数字，表示图像的像素宽度，最大值是 188，默认值为 88。

② <height>：数字，表示图像的像素高度。最大值是 400，默认值为 31。

③ <description>：包含文本，在呈现时可以作为围绕着该图像形成的链接元素的 title 属性。

此外还可以使用许多其他可选的频道元素。多数都是不言自明的：

① <language>：en-us

② <copyright>：Copyright 2003, James Lewin

③ <managingEditor>：dan@spam_me.com (Dan Deletekey)

④ <webMaster>：dan@spam_me.com (Dan Deletekey)

⑤ <pubDate>：Sat, 15 Nov 2003 0:00:01 GMT

⑥ <lastBuildDate>：Sat, 15 Nov 2003 0:00:01 GMT

⑦ <category>：ebusiness

⑧ <generator>：Your CMS 2.0

⑨ <docs>：http://blogs.law.harvard.edu/tech/rss

⑩ <cloud>：允许进程注册为“cloud”，当频道更新时通知它，为 RSS 提要且实现了一种轻量级的发布——订阅协议。

⑪ <ttl>：存活时间是一个数字，表示提要在刷新之前缓冲的分钟数。

⑫ <rating>：关于该频道的 PICS 评价。

⑬ <textInput>：定义可与频道一起显示的输入框。

⑭ <skipHours>：告诉聚集器哪些小时的更新可以忽略。

⑮ <skipDays>：告诉聚集器哪一天的更新可以忽略。

（2）项：项通常是提要中最重要的部分。每个项都可以关联某个 weblog、完整文档、电影评论、分类广告或者任何希望与频道连锁的内容和记录。通常频道中的其他元素可能不变，但项经常发生变化。

程序员可以有任意多个项。以前的规范限值为 15 个项，如果要保持向后兼容这仍然是一个很好的上限。

新闻项的元素，每个项通常包含三个元素：

① <title>：这是项的名称，在标准应用中被转换成 HTML 中的标题。

② <link>：这是该项的 URL。title 通常作为一个链接，指向包含在<link>元素中的 URL。

③ <description>：通常作为 link 中所指向的 URL 的摘要或者补充。并且所有的元素都是可选的，但是一个项至少要么包含一个<title>，要么包含一个<description>。

项还有其他一些可选的元素：

① <author>：作者的 e-mail 地址。

② <category>：支持有组织的记录。

③ <comments>：关于项的注释页的 URL。

④ <enclosure>：支持和该项有关的媒体对象。

⑤ <guid>：唯一与该项联系在一起的永久性链接。

⑥ <pubDate>：该项是什么时候发布的。

⑦ <source>：该项来自哪个 RSS 频道。当把该项聚合在一起时非常有用。

程序清单 5-25 是一个 RSS 2.0 文件的例子，说明了项和图像如何包含在频道中，并且所示的元素都是最常用的频道子元素。

程序清单 5-25：

```
<?xml version="1.0"?>
<rss version="2.0">
    <channel>
    <title>The channel's name goes here</title>
    <link>http://www.urlofthechannel.com/</link>
    <description>This channel is an example channel for an article.
    </description>
    <language>en-us</language>
```

```
    <image>
      <title>The image title goes here</title>
      <url>http://www.urlofthechannel.com/images/logo.gif</url>
      <link>http://www.urlofthechannel.com/</link>
    </image>
    <item>
      <title>The Future of content</title>
      <link>http://www.itworld.com/nl/ecom in act/11122003/</link>
      <description> The issue of people distributing and reusing
      digital media is a problem for many businesses. It may also be
      a hidden opportunity. Just as open source licensing has opened
      up new possibilities in the world of technology, it promises to do
      the same in the area of creative content.</description>
    </item>
    <item>
      <title>Online Music Services - Better than free?</title>
      <link>http://www.itworld.com/nl/ecom in act/08202003/</link>
      <description>More people than ever are downloading music from
      the Internet. Many use person-to-person file sharing programs like
      Kazaa to share and download music in MP3 format, paying nothing.
      This has made it difficult for companies to setup online music
      businesses. How can companies compete against free?</description>
    </item>
  </channel>
</rss>
```

5.6 面向服务的软件开技术

面向服务是当前发展和应用的热点，前面相关章节已经概述和分析了面向服务方法和技术，下面进一步总结、分析和概述面向服务技术。相关这方面书籍也特别多，本小节从易理解性、应用角度总结、分析服务技术。

5.6.1 Web 服务技术

通常 Web 服务技术是由服务提供者、服务消费者、WSDL（Web Service Definition Language）、SOAP（Simple Object Access Protocol）、UDD I （Universal Description, Discovery, and Integration），以及 WS 相关规范构成。使用这些 Web 服务技术，应用程序可以与平台和编程语言无关的方式相互通信。Web 服务是一个软件接口，它描述了一组可以在网络上通过标准化的 XML 消息传递访问的操作；它使用基于 XML 语言的协议来描述要执行的操作或者要与另一个 Web 服务交换的数据，图 5-17 所示是 Web 服务标准，图 5-18 所示是 Web 体系结构。

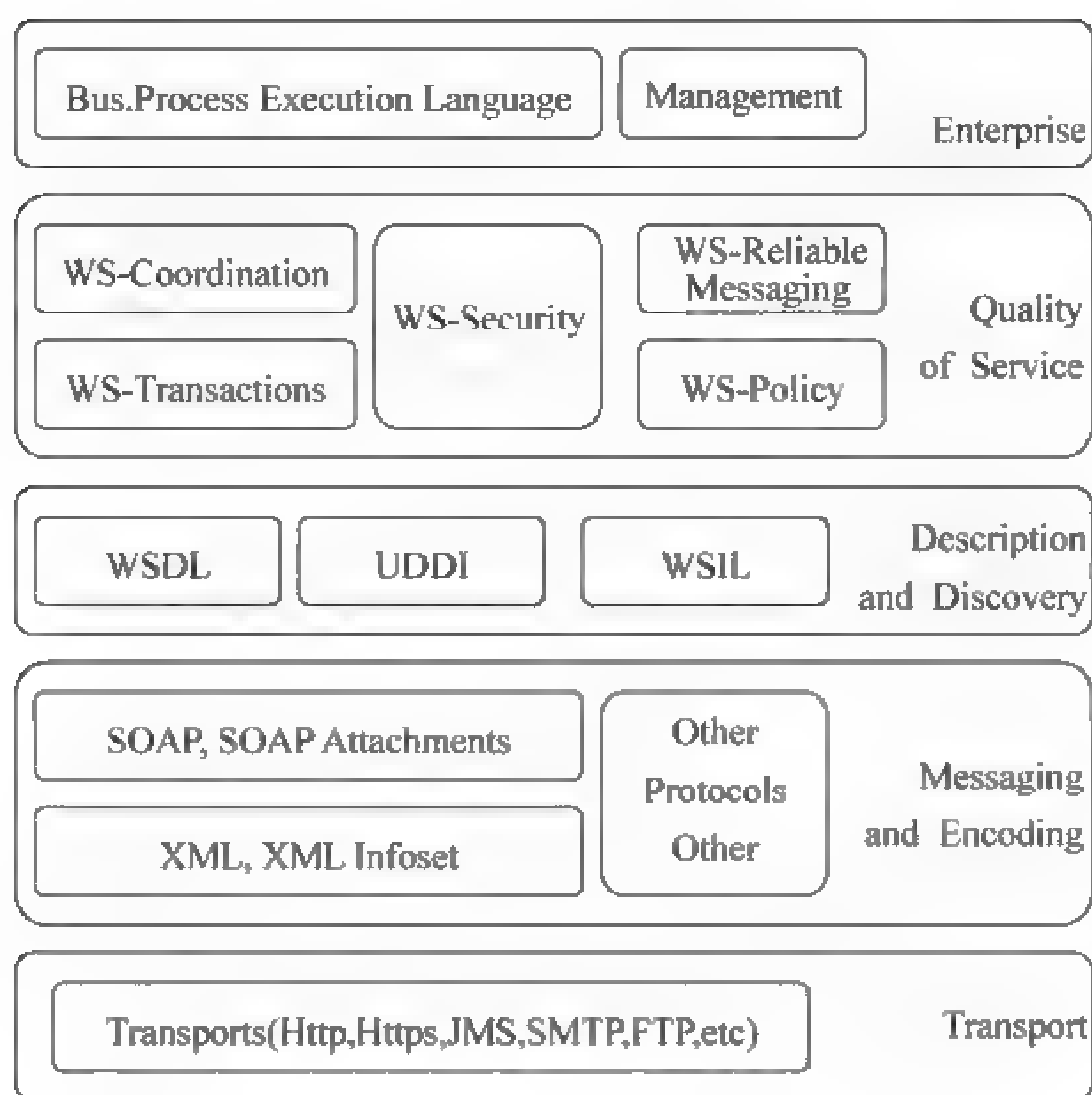

图 5-17　Web 服务标准

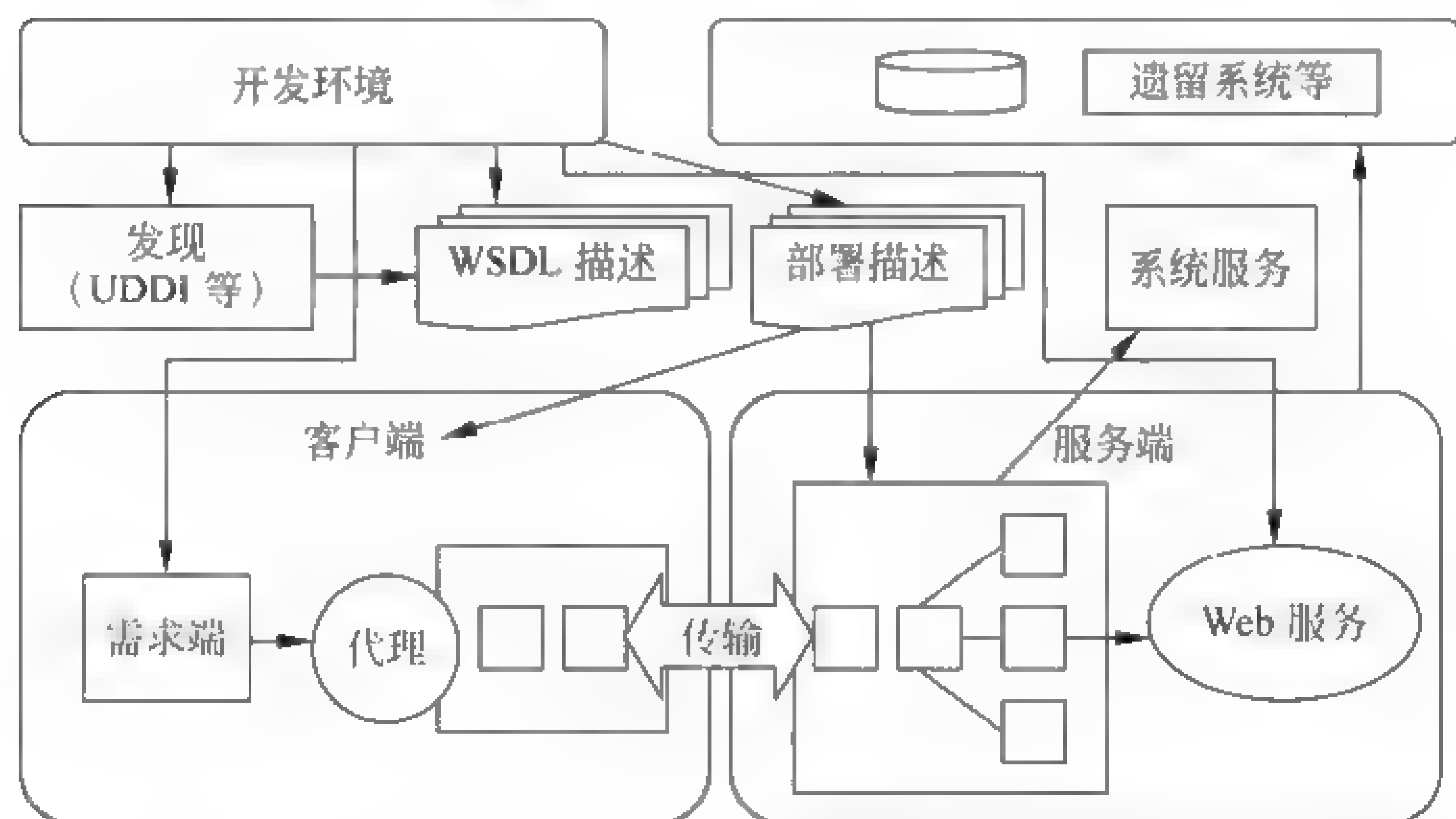

图 5-18　Web 服务体系结构

实际上，Web Service 最基本的组成部分为服务的提供者（Service Provider）和服务的请求者（Service Requester）。这看起来很像 C/S 架构的软件，与之不同的是，Web Service 两端的应用是通过基于标准的 XML 格式的协议进行通信的，这种最常用的协议就是 SOAP。当然，Web services 不简单地只是按这种方式进行通信。

按照 Web Service 的相关标准描述，服务的提供者应该首先通过 WSDL 和 UDDI 发布服务到一个统一的注册中心，且将这些提供服务信息存储到存储库中去。这样，服务的请求者就可以通过 WSDL 和 UDDI 发现到服务提供者所提供的服务，并可以通过应用的调用方法来使用这些服务。从而使得 Web 服务使用 XML 来描述任何（所有）数据，并使得这些数据能在跨平台上实现系统的数据交换。而且，Web 服务可以在较抽象的层面上工作，较抽象层面也可以按照需要动态地重新评估、修改或处理数据类型。所以，从技术层面上讲，Web 服务可以更方便地处理数据，并且允许软件更自由地进行通信。

1. Web服务组成技术概述

Web服务采用一系列相关协议来描述、传递服务和与服务交互。根据其通常的功能和使用，可以将这一系列协议进一步划分为组：

（1）第一组处理消息传递、接口描述、寻址和交付的问题。最有名的是消息传递协议，称为简单对象访问协议（SOAP）。此协议对消息进行了编码，这样就可以通过传输协议（如HTTP、IIOP、SMTP或其他协议）在网络上传递它们。Web 服务描述语言（WSDL）表示为一系列XML 语句，这些语句组成了每个服务的接口的定义。其中之一的规范是Web服务寻址（WS-Addressing），它定义了如何在分布式体系结构中唯一地进行 Web 服务寻址和标识Web服务。另一个流行的规范是Web服务调用框架（Web Services Invocation Framework），在这种框架中，程序员可以定义任何类型的组件的 WSDL 接口，即使它们没有使用相同的消息传递协议也可以在此框架中定义WSDL接口。

（2）第二组协议和规范定义了服务如何公开它们自己，以及如何在网络上相互发现。对于要相互查找的服务，UDDI为查找和访问服务定义了注册中心和相关协议。而Web服务检查语言（Web Services Inspection Language）是UDDI在不使用注册中心的情况下采用的一种可选机制。

（3）第三组是用于Web服务的安全性协议，它是从Web服务安全性（WS-Security）规范开始的，该规范为安全通信定义了基于权标的体系结构。以此为基础，有几个主要的组成规范：

① Web服务策略（WS-Policy）和相关的规范，定义了关于服务交互方式的策略规则。

② Web服务信任（WS-Trust），定义了安全交换的信任模型。

③ Web服务隐私（WS-Privacy），定义了如何维护信息的隐私。

④ Web 服务安全会话（WS-Secure Conversation），定义了如何使用在 Web 服务策略（WS-Policy）、Web服务信任（WS-Trust）和Web服务隐私（WS-Privacy）中定义的规则，以在用于交换数据的服务之间建立安全会话。

⑤ Web服务联盟（WS-Federation），定义了分布式标识的规则以及如何对其进行管理。

⑥ Web服务授权（WS-Authorization），定义了如何处理对访问和交换数据的授权。

⑦ WS-Reliability，一个来自OASIS的标准协议，用来提供可信赖的WEB服务间消息传递。

⑧ WS-ReliableMessaging，是一个提供信赖消息的协议，由Microsoft、BEA 和IBM发布，目前OASIS正对其实施标准化工作。

除了安全性模型之外，还有特定于应用程序的规范，其中包括Web服务的业务流程执行语言（Business Process Execution Language for Web Services，BPEL4WS），它定义了一起进行分布式事务处理的工作流操作、Web 服务事务（WS-Transaction）、We服务协调（WS-Coordination）。

Web服务主要是技术的集成，不过，它本身是独立于形式的。如前所述，组成Web服务的技术通常是用 XML 进行定义和交互的。然而，由于 XML 本身是一种独立的语言，所以Web服务也是独立的。因此，可以用许多编程语言（其中包括Java、Python、Perl、C#、Basic等等）来开发Web服务。这样通过提供更简单的统一接口，Web服务有助于改进用于移动环境和可移植环境的普及计算模型的工作方式。移动计算软件也很快采用Web服务通信模型，而这有助于改进可视化Web服务的接口问题。

2. WSDL

发布于 2001 年初的 WSDL 1.1 技术上已经被 2007 年发布的 W3C WSDL 2.0 推荐标准所替代。WSDL 2.0 提供了比 WSDL 1.1 更清晰的结构，并且更加灵活。但目前仍使用 WSDL 1.1 作为业务逻辑的描述，WSDL 2.0 未能广泛使用。

WSDL 提供了一种语法，用于将服务描述成一组交换消息的端点，它的特点主要包括：

（1）WSDL 文档充当一个或多个服务的（XML）描述，这种描述是与语言和平台无关的。

（2）WSDL 文档描述了这些服务、如何访问它们及期望的响应类型（如果有的话）。

（3）WSDL 描述了服务的类型、消息、操作、portType、位置和协议绑定。使用 WSDL 将 Web 服务描述成一组对消息进行操作的端点。WSDL 可以描述面向文档信息，也可以描述面向过程的信息。图 5-19 所示是 WSDL 协议的结构。

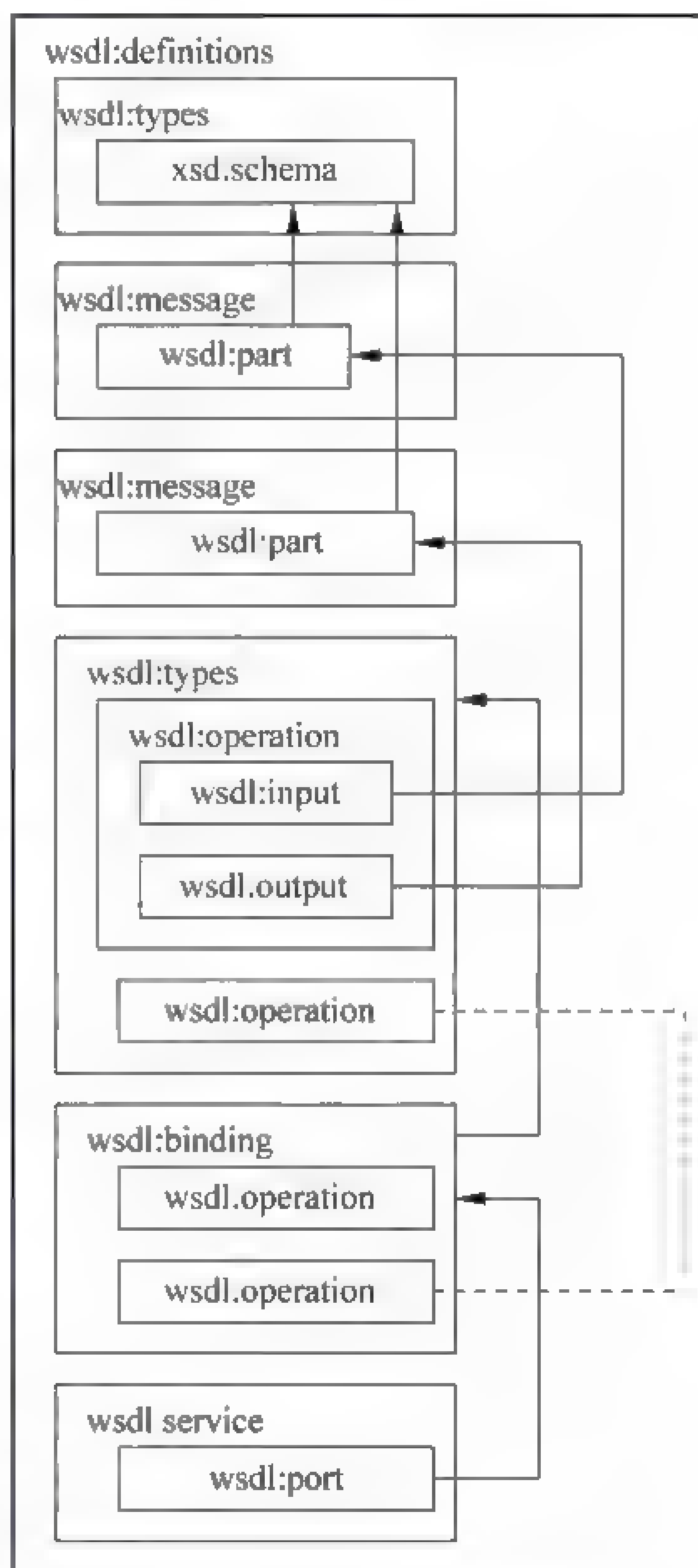

图 5-19　WSDL 协议结构

WSDL 协议主要包括以下六点：

（1）portType：portType 描述 Web 服务所提供的操作。它类似于一个 Java 接口，因为它描述了一组操作，它将消息元素合并成操作。

（2）message：message 是一个数据元素。操作用它来传送该操作的数据，并通过列出所交换的数据类型来描述客户机和服务之间的通信。

（3）types：types 元素中描述了类型，通常用 XML Schema 完成描述，types 类似于 Java 类和基本类型。

（4）operation：operation 就像一个 Java 方法，它包含了传入消息、传出消息和出错消息。程序员可以将操作的传入消息当作是 Java 编程语言中方法的参数。可以将操作的传出消息当作 Java 编程语言中方法的返回类型，可以将出错消息当作 Java 异常。

（5）binding：binding 将 portType 绑定到特定的协议（例如 SOAP 1.1、HTTP GET/POST 或 MIME）。

（6）service：service 定义了某个特定绑定（binding）的连接信息，它可以有一个或多个端口，每个端口都定义一个不同的连接方法（例如 HTTP/SMTP 等等）。

其实，WSDL 1.1 文档使用一个固定的根元素，一般命名为<wsdl:definitions>。在根元素中，WSDL 1.1 命名空间定义了一个来引用不同的 WSDL 1.1 文档，而子元素和六个元素就是实际的服务描述：

（1）<wsdl:import>引用另一个 WSDL 1.1 文档，将其描述加到本文档中。

（2）<wsdl:types>定义消息交换所使用的 XML 类型和元素。

（3）<wsdl:message>定义一个实际的消息，包含 XML 类型或元素。

（4）<wsdl:portType>定义一个服务所实现的操作抽象集。

（5）<wsdl:binding>定义<wsdl:portType>的一个使用特定协议和格式的具体实现。

（6）<wsdl:service>定义一个服务整体，包括一个或多个包含<wsdl:binding>元素访问信息的<wsdl:port>元素。

另外还有一个可用于文档记录的<wsdl:document>元素，它可以是<wsdl:definitions>元素的第一个子元素，也可以是上面其他元素的第一个子元素。

一个完整的服务描述一般至少包含一个除 <wsdl:import> 元素之外的其他元素，但是所有这些元素不一定要位于同一个文档中。程序员可以使用 <wsdl:import> 元素将多个文档整合成为一个完整的 WSDL 描述，这使程序员能够灵活分割描述来满足要求。例如，前三个描述元素（<wsdl:types>、<wsdl:message> 和 <wsdl:portType>）一起构成了完整的服务接口描述（可能是由一个体系结构团队定义的），所以将它们保存到不同的面向实现的<wsdl:binding>和<wsdl:service>元素上是很有意义的。所谓以的主流 Web 服务协议都支持将描述分割到多个 WSDL 文档中。

程序清单 5-26 和程序清单 5-27 是一个分割成两个 WSDL 文档的 WSDL 服务描述示例，其中接口描述组件位于 BookServerInterface.wsdl 文件，而实现组件则位于 BookServerImpl.wsdl。这个 WSDL 网上书店的一部分 WSDL 文件。

程序清单 5-26 （BookServerInterface.wsdl）：

```
<wsdl:definitions ... xmlns:tns="http://sosnoski.com/ws/library/
BookServerInterface"
    targetNamespace="http://sosnoski.com/ws/library/BookServerInterface">
  <wsdl:document>Book service interface definition.</wsdl:document>
  <wsdl:types>
    <xs:schema ...
        targetNamespace="http://sosnoski.com/ws/library/BookServerInterface">
      <xs:import namespace="http://sosnoski.com/ws/library/types"
          schemaLocation="book-types.xsd"/>
      ...
    </xs:schema>
  </wsdl:types>
  <wsdl:message name="getBookMessage">
    <wsdl:part name="part" element="tns:getBook"/>
  </wsdl:message>
  <wsdl:message name="getBookResponseMessage">
    <wsdl:part name="part" element="tns:getBookResponse"/>
  </wsdl:message>
  ...
  <wsdl:message name="addBookMessage">
    <wsdl:part name="part" element="tns:addBook"/>
  </wsdl:message>
  <wsdl:message name="addBookResponseMessage">
    <wsdl:part name="part" element="tns:addBookResponse"/>
  </wsdl:message>
  <wsdl:message name="addDuplicateFault">
```

```
    <wsdl:part name "fault" element "tns:addDuplicate"/>
  </wsdl:message>
  <wsdl:portType name="BookServerPortType">
    <wsdl:documentation>
      ......
    </wsdl:documentation>
    <wsdl:operation name="getBook">
      <wsdl:documentation>
        Get the book with a particular ISBN.
      </wsdl:documentation>
      <wsdl:input message="tns:getBookMessage"/>
      <wsdl:output message="tns:getBookResponseMessage"/>
    </wsdl:operation>
    ...
    <wsdl:operation name="addBook">
      <wsdl:documentation>Add a new book.</wsdl:documentation>
      <wsdl:input message="tns:addBookMessage"/>
      <wsdl:output message="tns:addBookResponseMessage"/>
      <wsdl:fault message="tns:addDuplicateFault" name="addDuplicateFault"/>
    </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```

程序清单 5-27 （BookServerImpl.wsdl）：

```
<wsdl:definitions ... xmlns:ins="http://sosnoski.com/ws/library/
BookServerInterface"
    xmlns:tns="http://sosnoski.com/ws/library/BookServer"
    targetNamespace="http://sosnoski.com/ws/library/BookServer">
  <wsdl:document>
    Definition of actual book service implementation.
  </wsdl:document>
  <wsdl:import namespace="http://sosnoski.com/ws/library/BookServerInterface"
      location="BookServerInterface.wsdl"/>
  <wsdl:binding name="BookServerBinding" type="ins:BookServerPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style=
     "document"/>
    <wsdl:operation name="getBook">
      <soap:operation soapAction="urn:getBook"/>
      <wsdl:input>
        <soap:body/>
      </wsdl:input>
      <wsdl:output>
        <soap:body/>
      </wsdl:output>
```

```
    </wsdl:operation>
    ...
    <wsdl:operation name="addBook">
      <soap:operation soapAction="urn:addBook"/>
      <wsdl:input>
        <soap:body/>
      </wsdl:input>
      <wsdl:output>
        <soap:body/>
      </wsdl:output>
      <wsdl:fault name="addDuplicateFault">
        <soap:fault name="addDuplicateFault"/>
      </wsdl:fault>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="BookServer">
    <wsdl:port name="BookServerPort" binding="tns:BookServerBinding">
      <soap:address location="http://localhost:8080/cxf/BookServer"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

WSDL 由以下五种元素组成：

（1）<types>元素。<types>元素允许程序员指定数据类型，无论这些类型多简单或多复杂，这些类型是 WSDL 文件所描述的 Web 服务接口必需的。XML Schema 是开放的，因此在 WSDL 中使用它使得 Web 服务定义不受到特定于编程语言的类型的约束，也不受到特定于供应商的类型的约束。这是因为 XML Schema 为 Web 服务提供了一种获得业界认可的方式，用来定义复杂数据类型以及预先确定好的简单数据类型（例如 string、date 和 numeric 等）。遵循 XML Schema 规则的用户定义的 XML 协议的描述通常位于 WSDL 文件中。其中：

① schema 元素是 XML Schema 文档的根元素。XML 协议的所有元素类型和属性类型的定义都在其中。

② element 元素允许程序员定义元素的类型。

③ complexType 元素定义了一个复杂类型。如果一个元素有属性和（或）子元素，那么就认为该元素的类型是复杂类型。且类型既有属性又有子元素。元素的子元素可以遵循三种模型：any、choice 和 sequence（相关 XML Schema 详细类型如 http://www.w3.org/TR/xmlschema-0/#CreatDt），如下程序片段就是 XML Schema 在<types> WSDL 的应用。

```
<wsdl:definitions
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://dvd.com"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns:tns="http://dvd.com">
  ...
```

```
 <wsdl:types>
  <xsd:schema>
......
  </xsd:schema>
 </wsdl:types>
 ...
</wsdl:definitions>
```

（2）<message>元素。<message>元素将数据（数据类型在<types>元素中进行定义）分组成一个用于逻辑网络传输的特征符，并将数据绑定到一个名称上。该名称用来在操作定义的上下文中引用<message>。<message>中的每个数据实例都在<part>元素中进行了声明。

```
<wsdl:definitions
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 ...>
  ...
 <wsdl:message name="">
  <wsdl:part name="" type="指出 WSDL 中的 type 数据"/>
  <wsdl:part name="until" type="指出 WSDL 中的 type 数据"/>
 </wsdl:message>
 ...
</wsdl:definitions>
```

（3）<portType>元素。<portType>元素在 WSDL 中相当于 Java 中的接口。它将对<message>元素的引用分组成逻辑操作，即一个进程可以对另一个进程执行这些操作，并将它们绑定到一个名称。<operation>可以包含<input>、<output>和<fault>元素。对于所有这三个元素，message 属性引用 WSDL 文件中所定义的<message>元素。<input>元素声明客户机向 Web 服务请求传输的需求。<output>声明 Web 服务响应的内容。<fault>元素描述当 Web 服务设法响应客户机的请求时所发生的任何消息级异常，程序片段如下。

```
<wsdl:definitions
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 ...>
...
 <wsdl:portType name="">
  <wsdl:operation name="">
   <wsdl:input message="" />
  </wsdl:operation>
 ...
</wsdl:definitions>
```

（4）<binding>元素。WSDL<binding>元素的内容将这些抽象的钩接点与 Web 协议束缚起来。binding 元素既有 name 属性（用来在 WSDL 文档内标识该绑定），又有 type 属性（用来引用该元素描述绑定时所针对的 portType），还有<operation>元素。并与它所绑定的<portType>中的每个<operation>元素对应。在<operation>元素也有<input>/<output>/<fault>元素，与相应

的<operation>元素中所定义的元素相对应。描述绑定的元素嵌套在<binding>元素的这些子元素之中，这是为了将消息传递协议的详细内容链接到目标 portType 中所提到的通则上。目前，W3C 推荐了三个 Web 服务的绑定：HTTP 上的 SOAP（SOAP over HTTP）、HTTP GET/POST 和 SOAP/MIME。以下是以 HTTP 上的 SOAP 为例，程序片段如下。

```
<wsdl:definitions>
 ...
 <wsdl:binding name=" " type= "">
  <wsdl:operation name=" ">
   <wsdl:input>
    <soap:body namespace="http://" use="literal"/>
   </wsdl:input>
   <wsdl:output>
    <soap:body namespace="http:// " use="literal"/>
   </wsdl:output>
    <wsdl:fault name=" ">
    <soap:fault namespace="http:// "  use="literal"/>
   </wsdl:fault>
...
 </wsdl:binding>
 ....
</wsdl:definitions>
```

在程序片段中，<soap:binding>元素表示 soap 通信的消息传递样式（rpc）和期望使用的传输协议（HTTP）。<soap:operation>元素的 soapAction 属性被转换成 HTTP 头，该头声明了将被发送的 HTTP 请求的目的。在 SOAP 1.1 中引入了这个头，以允许防火墙和其他预处理节点对 SOAP 请求进行过滤。在 SOAP 1.2 中，不使用这个头，而建议将请求中的请求目的声明为一个参数，称作 action。因而通常就让这个头空着。<soap:body>和<soap:fault>元素表示消息数据应当放到所生成的 SOAP <Envelope>中的什么地方。在上面的案例中，除了 RentDVD 出错数据应当包含在 SOAP <Fault>元素中之外，其他所有消息都将包含在常规的 SOAP <Body>元素中。

（5）<service>元素。<service> WSDL 元素将某个具体的绑定与网络上的一个或多个进程相关联，这些进程可以根据绑定所实现的 portType 来处理请求。对于 HTTP 上的 SOAP，这就是指向那个进程的 URL，程序片段如下：

```
<wsdl:service name="">
 <wsdl:documentation> </wsdl:documentation>
 <wsdl:port name="" binding="">
  <soap:address location="http://" />
 </wsdl:port>
 <wsdl:port name="" binding="">
  <soap:address location="" />
 </wsdl:port>
</wsdl:service>
```

3．SOAP

简单对象访问协议（SOAP）是一种标准化的通信规范，主要用于 Web 服务中。SOAP 的出现是为了简化网页服务器（Web Server）在从 XML 数据库中提取数据时，无须花时间去格式化页面，并能够让不同应用程序之间透过 HTTP 通信协定，以 XML 格式互相交换彼此的数据，使其与编程语言、平台和硬件无关。此标准由 IBM、Microsoft、UserLand 和 DevelopMentor 在 1998 年共同提出，并得到 IBM，Lotus，Compaq 等公司的支持，于 2000 年提交给万维网联盟（World Wide Web Consortium；W3C），目前 SOAP 1.1 版是业界共同的标准，属于第二代的 XML 协定（第一代具主要代表性的技术为 XML-RPC 以及 WDDX）。

SOAP 是在分散或分布式的环境中交换信息的简单的协议，是一个基于 XML 的协议。它包括四个部分：

（1）SOAP 封装（envelop）：封装定义了一个描述消息中的内容是什么，是谁发送的，谁应当接受并处理它，以及如何处理它们的框架。

（2）SOAP 编码规则（encoding rules）：用于表示应用程序需要使用的数据类型的实例。

（3）SOAP RPC（RPC representation）：表示远程过程调用和应答的协定。

（4）SOAP 绑定（binding）：使用底层协议交换信息。

SOAP 通常采用已经广泛使用的两个协议：HTTP 和 XML。其中 HTTP 用于实现 SOAP 的 RPC 风格的传输，而 XML 是它的编码模式，一个 SOAP 请求实际上就是一个 HTTP POST 请求。图 5-20 所示是 SOAP 结构，程序清单 5-28 是 SOAP 描述结构，程序清单 5-29 和程序清单 5-30 是 SOAP 请求和响应的例子。

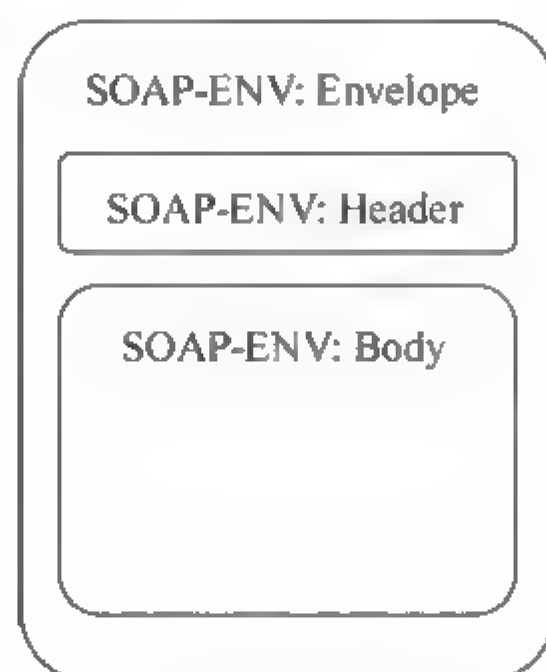

图 5-20　SOAP 结构

程序清单 5-28 （SOAP 程序结构）：

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Header>
  ......
</soap:Header>

<soap:Body>
```

```
  ......
  <soap:Fault>
    ......
  </soap:Fault>
</soap:Body>
</soap:Envelope>
```

程序清单 5-29 （SOAP 请求）：

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://localhost:8080/axis2/services/MyService/">
    <m:GetStockPrice>
      <m:StockName> </m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

程序清单 5-30 （SOAP 响应）：

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://localhost:8080/axis2/services/MyService/">
    <m:GetStockPriceResponse>
      <m:Price>34.5</m:Price>
    </m:GetStockPriceResponse>
  </soap:Body>
</soap:Envelope>
```

SOAP 由以下六种元素组成：

（1）SOAP 信封。Web 服务消息的基本单元是实际的 SOAP 信封，这是包含处理消息所必需的所有信息的 XML 文档，如下程序清单，它获得了一个简单的 Envelope，其命名空间指定为 SOAP 1.2 版本，其中包含两个子元素 Header 和 Body。

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/SOAP-envelope">
 <env:Header>
</env:Header>
<env:Body>
</env:Body>
</env:Envelope>
```

（2）SOAP Header。SOAP 消息中的 Header 用于提供有关消息本身的信息，与用于应用

程序的信息相对。如下程序清单有一个 WS-Addressing 元素，其中包含有关消息将送达何处，以及应将应答送达何处的信息。Header 可包含关于消息本身的所有类型的消息。规范考虑了 SOAP 消息在送达最终目的地的过程中可能实际由多个中间层处理的情况，这也很清楚地说明了中间层应如何对待在 Header 中找到的信息。

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/SOAP-envelope">
 <env:Header>
  <wsa:ReplyTo xmlns:wsa=
        "http://schemas.xmlSOAP.org/ws/2004/08/addressing">
   <wsa:Address>
http://schemas.xmlSOAP.org/ws/2004/08/addressing/role/anonymous
   </wsa:Address>
  </wsa:ReplyTo>
  <wsa:From>
   <wsa:Address>
http://localhost:8080/axis2/services/MyService</wsa:Address>
  </wsa:From>
  <wsa:MessageID>ECE5B3F187F29D28BC11433905662036</wsa:MessageID>
  </env:Header>
<env:Body>
</env:Body>
</env:Envelope>
```

（3）SOAP 信息。发送 SOAP 消息时，都是有目的性的，即将请求信息交互到对应的对接点，这些信息放在 Envelope 的 Body 中，程序清单如下：

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/SOAP-envelope">
 <env:Header>
...
</env:Header>
 <env:Body>
  ......
 </env:Body>
</env:Envelope>
```

（4）SOAP 样式和编码。在创建应用程序时，将需要确定要发送和接收的实际有效信息的结构，为此，这时就需要样式和编码来完成。目前有两种不同的主流 SOA 消息编程样式。第一种是 RPC 样式，基于 SOAP 消息创建远程过程调用（Remote Procedure Call）的概念，如下程序清单就是 RPC 使用样式。

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/SOAP-envelope">
 <env:Header>
```

（5）SOAP 消息交换模式。可以发送请求并等待响应，发送请求但不等待响应，发送请

求并在到达最终的目的地前通过多个中间层。目前只有两个选择。

① 请求/响应：在请求/响应模式中，以 SOAP 消息的形式发送请求，然后直接等待发送回响应。请求可以为同步的，也可以是异步的。

② 单向消息传递：这种情况又称为"Fire and Forget"方法，发送请求但并不等待响应。可以在仅传递信息时或并不关心接收者对此如何响应时使用此方法。

（6）创建 SOAP 请求与响应。最初出现用于使用 SOAP 消息的 Java API 时，其用途非常特定化。它们的用途相当直接，用于创建 SOAP 消息、Envelope、Header、Body 等。当要直接创建 SOAPEnvelope、SOAPBody 等内容时，可以向消息体添加 echo 和 category 等元素。并将从其中创建连接进行调用，同样也能遍历 SOAP 消息的结构来获取实际的内容，程序清单如下：

```
REQUEST: /* SOAP 请求*/
<SOAPenv:Envelope xmlns:SOAPenv=
       "http://schemas.xmlSOAP.org/SOAP/envelope/"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <SOAPenv:Body>
  <req:echo xmlns:req=
        "http://localhost:8080/axis2/services/MyService/">
         /* 采用开源软件 axis2 的服务*/
     <req:category>classifieds</req:category>
  </req:echo>
 </SOAPenv:Body>
</SOAPenv:Envelope>

RESPONSE: /* SOAP 响应*/
<SOAPenv:Envelope xmlns:SOAPenv=
      "http://schemas.xmlSOAP.org/SOAP/envelope/" xmlns:wsa=
      "http://schemas.xmlSOAP.org/ws/2004/08/addressing">
 <SOAPenv:Header>
  <wsa:ReplyTo> /* WS-Addressing*/
   <wsa:Address>
http://schemas.xmlSOAP.org/ws/2004/08/addressing/role/anonymous
   </wsa:Address>
  </wsa:ReplyTo>
  <wsa:From>
   <wsa:Address>
http://localhost:8080/axis2/services/MyService</wsa:Address>
  </wsa:From>
  <wsa:MessageID>ECE5B3F187F29D28BC11433905662036</wsa:MessageID>
 </SOAPenv:Header>
 <SOAPenv:Body>
```

```
    <req:echo xmlns:req
          "http://localhost:8080/axis2/services/MyService/">
       <req:category>classifieds</req:category>
    </req:echo>
   </SOAPenv:Body>
  </SOAPenv:Envelope>
```

4．UDDI

UDDI 是统一描述、发现和集成（Universal Description, Discovery, and Integration）的缩写。它是一个基于 XML 的跨平台的描述规范，可以使世界范围内的企业在互联网上发布自己所提供的服务。UDDI 是核心的 Web 服务标准之一，它通过 SOAP 进行消息传输，用 WSDL 描述 Web 服务及其接口使用。图 5-21 是 UDDI 与 SOAP 结构图。在图中，UDDI 消息的传输，通过 HTTP 从客户机的 SOAP 请求传到注册中心结点，再反向传输。注册中心服务器的 SOAP 服务器接收 UDDI SOAP 消息、进行处理，然后把 SOAP 响应返回给客户机。就注册中心条例而言，客户机发出的要修改数据的请求必须确保是安全的、经过验证的事务。

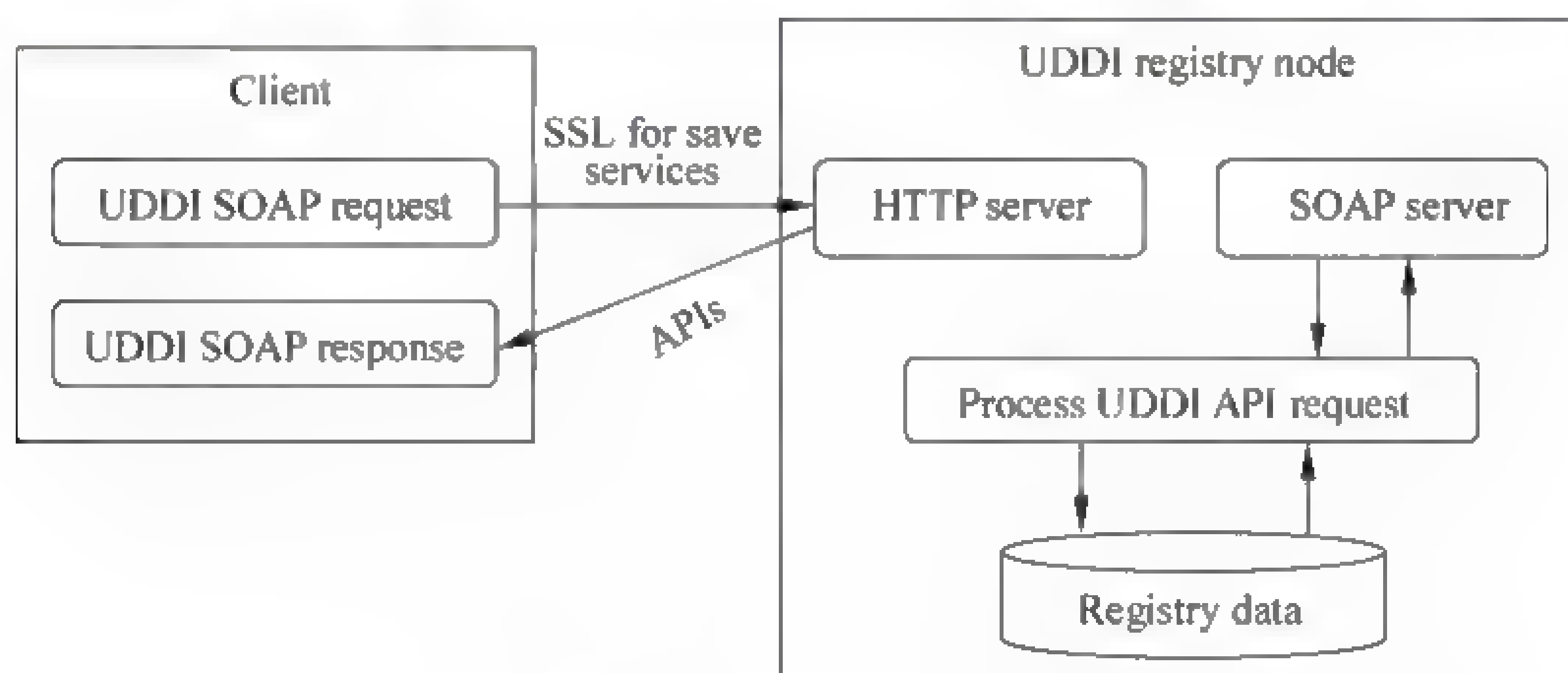

图 5-21　UDDI 与 SOAP 结构

1）UDDI 基本概念

（1）UDDI 规范：UDDI 规范版包括两个规范文本，UDDI Programmer's API（UDDI 程序员 API 规范）和 UDDI Data Structure Reference（UDDI 数据结构参考）。前者定义了 UDDI Operator Site 能够支持的 API 接口，而后者则描述了在 API 中具体 XML 描述的数据结构的具体定义。UDDI 规范是 UDDI Operator Site 是实现蓝本，也是需要访问 UDDI Registry 的 Web 服务的参考规范。

（2）UDDI Registry（UDDI 注册中心）：UDDI Registry 是所有提供公共 UDDI 注册服务的站点的通称。UDDI Registry 是一个逻辑上的统一体，在物理上则是以分布式系统的架构实施的，而不同站点之间是采用 P2P（对等网络）架构实施的，因此访问其中任意一个站点就基本等于访问了 UDDI Registry。

（3）UDDI Operator Site（UDDI 注册中心操作入口站点，简称 UDDI 操作入口）：UDDI Operator Site 是 UDDI Registry 中每一个对等结点，对于 UDDI Operator Site 的查询所获得的结果是覆盖全 UDDI Registry 中的信息，信息查询无须身份认证；而在 UDDI Operator Site 上

进行信息发布则必须使用该 UDDI Operator Site 自身的用户方能实施，同时以后的更新、删除都必须通过这个 Operator Site，并使用初始发布时使用的用户进行权限认证。

（4）Compatible UDDI Registry（兼容的 UDDI 注册中心）：所有兼容 UDDI 规范但并非属于提供公共服务的 UDDI Registry 的个别 UDDI 注册中心，都称为兼容的 UDDI 注册中心。

2）UDDI 信息模型

UDDI 注册使用的核心信息模型由 XML Schema 定义。使用 XML 是因为它提供了平台无关的数据描述并很自然的描述了数据的层次关系。而选择 XML Schema 是因为它支持丰富的数据类型，便捷的描述方式及其按信息模型对数据进行验证的能力。UDDI XML Schema 定义了四种主要信息类型（它们是技术人员在需要使用合作伙伴所提供的 Web 服务时必须了解的技术信息）：商业实体信息（businessEntity）、服务信息（businessService）、绑定信息（bindingTemplate）、服务调用规范（tModel）的说明信息。图 5-22 所示是 UDDI 信息模型。

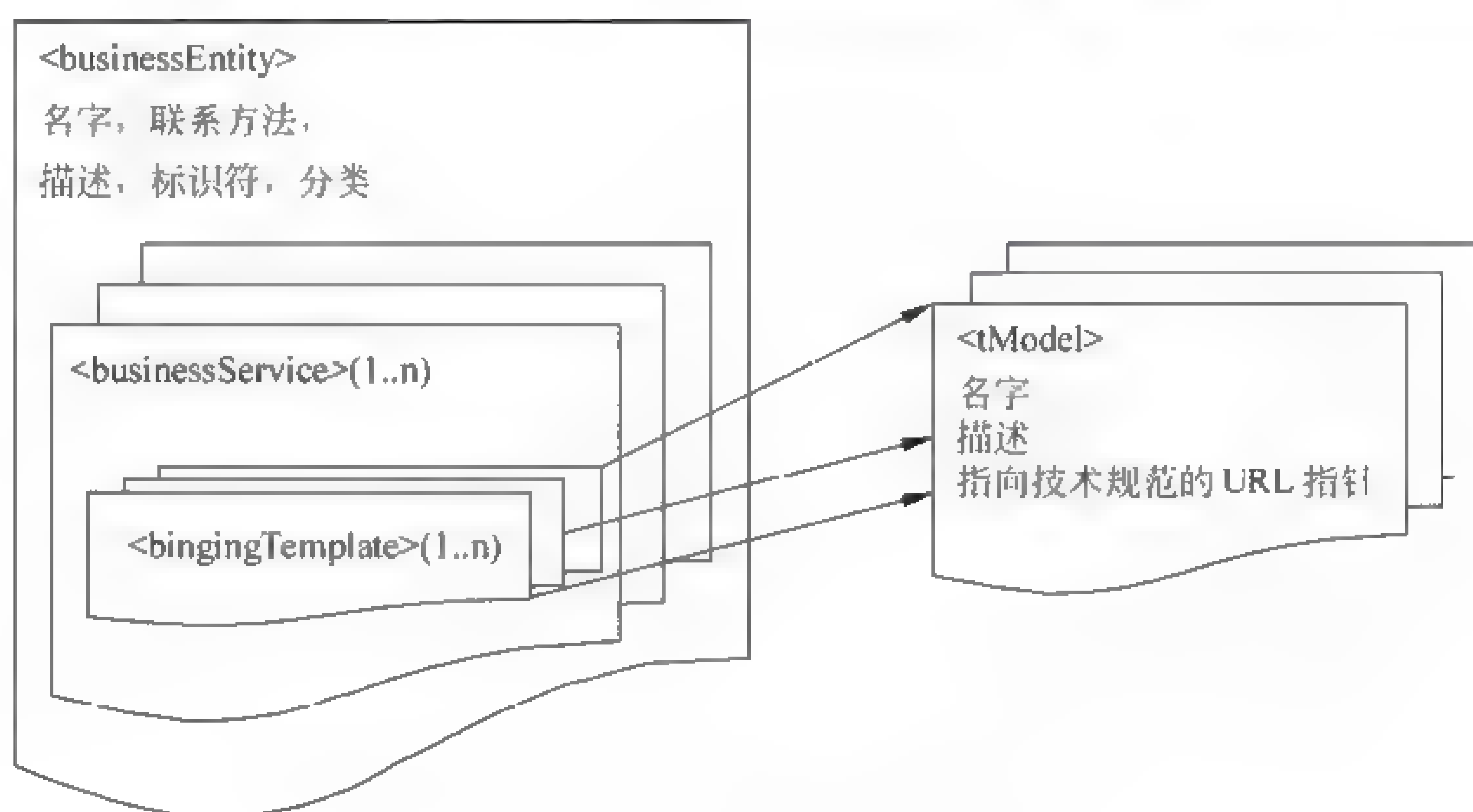

图 5-22　UDDI 信息模型

四种主要信息类型概述如下：

（1）tModel。tModel 是一个技术规范的超类，tModel 能够描述商业标识符数据库、分类方法、技术规范、网络协议等各类的技术规范。图 5-23 所示是 tModel 的结构。在 UDDI 的数据模型中，tModel 是一个很特殊的数据项。tModel 描述了一切技术信息，tModel 的全体组成了 UDDI 中的所有技术注册信息。

当一个程序或是程序员需要调用某个特定的 Web 服务时，必须根据应用要求得到了足够充分的调用规范等相关信息，以使调用被正确地执行。因此，每一个 bindingTemplate 元素都包含一个特殊的元素，该元素包含了一个列表，列表的每个子元素分别是一个调用规范的引用。这些引用作为一个标识符的杂凑集合，并组成了类似指纹的技术标识，用来查找、识别实现了给定行为或编程接口的 Web 服务。实际上，这些引用是访问服务所需要的关键的调用规范信息。被称为 tModel 的数据项是关于调用规范的元数据，它包括服务名称，发布服务的组织以及指向这些规范本身的 URL 指针等，如下程序片段是 tModel 结构的 XML Schema 定义。

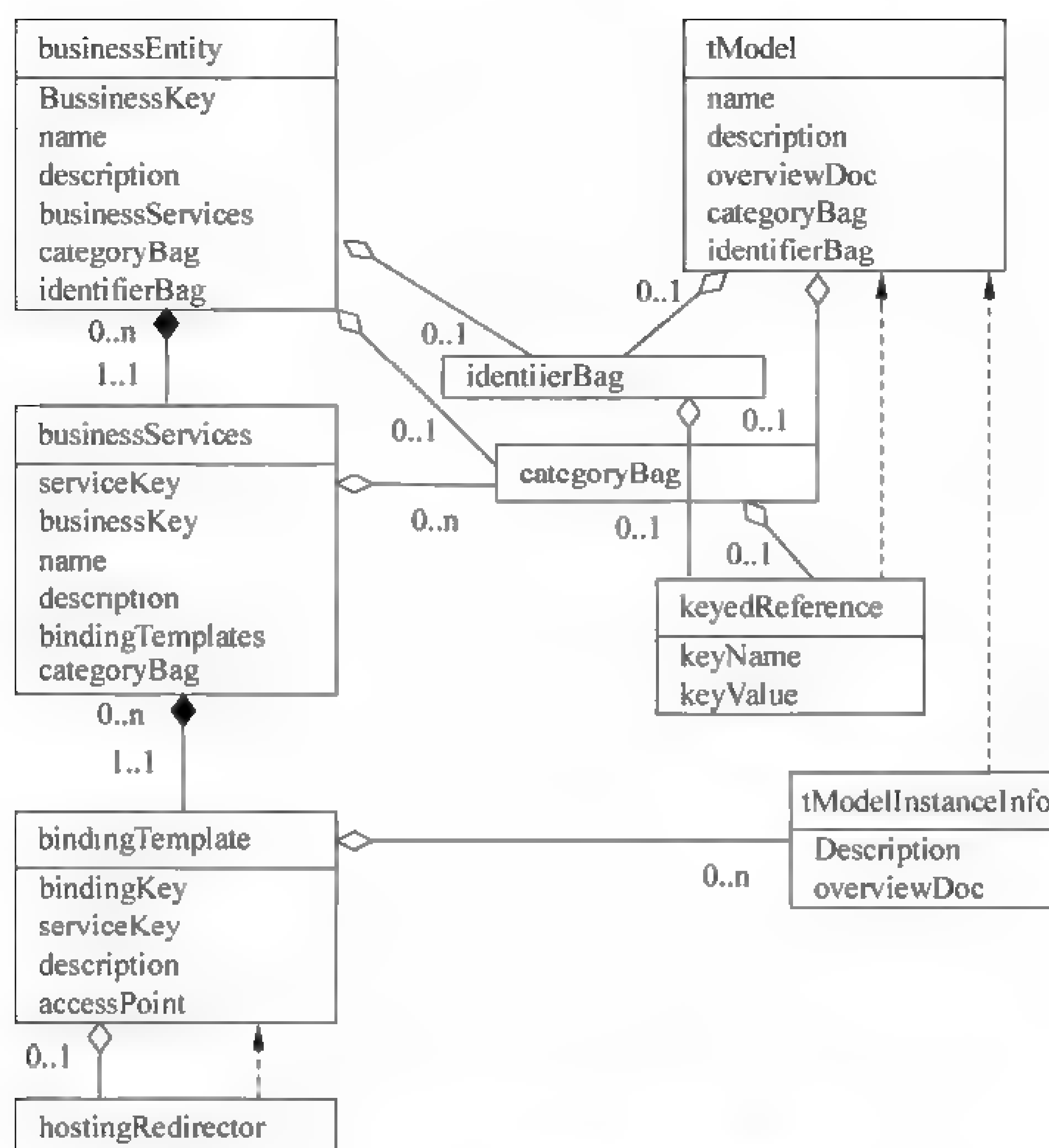

图 5-23　UDDI 数据模型关系图

```
<element name = "tModel">
<type content = "elementOnly">
<group order = "seq">
<element ref = "name"/>
<element ref = "description" minOccurs = "0" maxOccurs = "*"/>
<element ref = "overviewDoc" minOccurs = "0" maxOccurs = "1"/>
<element ref = "identifierBag" minOccurs = "0" maxOccurs = "1"/>
<element ref = "categoryBag" minOccurs = "0" maxOccurs = "1"/>
</group>
<attribute name = "tModelKey" minOccurs = "1" type = "string"/>
<attribute name = "operator" type = "string"/>
<attribute name = "authorizedName" type = "string"/>
</type>
</element>
```

其中，overviewDoc 包含的是规范的关键信心，包含了一系列的 URL，通过这些 URL 可以访问到这个 tModel 的具体技术规范文本。tModelKey、operator 和 authorizedName 这三个 tModel 的直接属性分别表示：tModel 的主键、实施注册的 UDDI 操作入口站点和对该 tModel 拥有所有权的用户 ID。一般的，tModelKey 是在注册后由 UDDI 注册中心自动赋予，并在 tModel 整个生命周期中有效。以后仅能通过 authorizedName 指定的用户 ID 在由 operator 指定的操作入口站点上进行该 tModel 的信息更新和对象删除，任意其他 ID 不能操纵本对象，

同时也不能在其他操作入口站点上对该实体对象的数据进行维护。

（2）businessEntity。支持对 UDDI 商业注册的商业信息发布和发现的核心 XML 元素都包含在 businessEntity 结构中，这个结构是商业实体专属信息集的最高层的数据容器，位于整个信息结构的最上层。所有 businessEntity 中的信息支持 UDDI 的黄页分类法。因此可以执行这样的搜索，如可以定位属于某个行业分类或提供某种产品的企业，也可以是定位处于某个地域范围内的企业。目前在 UDDI 中内置的分类法包括 NAICS 工业分类法、UN/SPSC 产品分类法等，如下程序片段是 businessEntity 结构的 XML Schema 定义。

```
<element name = "businessEntity">
<type content = "elementOnly">
<group order = "seq">
<element ref = "discoveryURLs" minOccurs = "0" maxOccurs = "1"/>
<element ref = "name"/>
<element ref = "description" minOccurs = "0" maxOccurs = "*"/>
<element ref = "contacts" minOccurs = "0" maxOccurs = "1"/>
<element ref = "businessServices" minOccurs = "0" maxOccurs = "1"/>
<element ref = "identifierBag" minOccurs = "0" maxOccurs = "1"/>
<element ref = "categoryBag" minOccurs = "0" maxOccurs = "1"/>
</group>
<attribute name = "businessKey" minOccurs = "1" type = "string"/>
<attribute name = "operator" type = "string"/>
<attribute name = "authorizedName" type = "string"/>
</type>
</element>
```

其中，name、description 分别表示该服务的名、描述等信息，categoryBag 的作用与 businessEntity 中是类似的。bindingTemplates 是一个 bindingTemplate 的容器，它表示了这个 businessService 所包含的所有技术绑定信息。在 bindingTemplates 中的每个 bindingTemplate 条目都是本 Web 服务的一条技术描述信息。这个结构表达了 businessService 和 bindingTemplate 的父子包含关系。serviceKey 和 businessKey 两个直接属性分别表示 businessService 的主键和 businessService 的父类容器 businessEntity 的主键标识。一般地，serviceKey 是在注册后由 UDDI 注册中心自动赋予，并在 businessService 整个生命周期中有效。而 businessKey 的值仅当 businessService 的父类容器发生变化时才会被修改。

（3）bindingTemplate。对于每一个 businessService，存在一个或多个 Web 服务的技术描述 bindingTemplate。这些技术描述包括应用程序连接远程 Web 服务并与之通信所必须的信息；而这些信息包括 Web 应用服务的地址、应用服务宿主和调用服务前必须调用的附加应用服务等，如下程序片段就是 bindingTemplate 结构的 XML Schema 定义。

```
<element name = "bindingTemplate">
<type content = "elementOnly">
<group order = "seq">
<element ref = "description" minOccurs = "0" maxOccurs = "*"/>
<group order = "choice">
```

```
<element ref - "accessPoint" minOccurs - "0" maxOccurs - "1"/>
<element ref = "hostingRedirector" minOccurs = "0" maxOccurs = "1"/>
</group>
<element ref = "tModelInstanceDetails"/>
</group>
<attribute name = "bindingKey" minOccurs = "1" type = "string"/>
<attribute name = "serviceKey" type = "string"/>
</type>
</element>
```

其中，accessPoint 定义了由这个 bindingTemplate 描述的服务调用入口的访问地址，用户可以通过这个地址访问具体的服务。而 hostingRedirector 则是因为 accessPoint 无法满足需求而提供的替代机制，同时 hostingRedirector 机制也是一种及为有效的满足当前电子交易市场、ISP 等中间机构存在的情况下的强有力的描述托管机制。tModelInstanceDetails 包含了这个 bindingTemplate 所定位的调用入口的调用规范的描述，这些描述都以 tModel 的形式出现。事实上，tModel 并不是 bindingTemplate 的子元素，这里应该理解成引用关系。bindingKey 和 serviceKey 两个直接属性分别表示 bindingTemplate 的主键和 bindingTemplate 的父类容器 businessService 的主键标识；一般地，bindingKey 是在注册后由 UDDI 注册中心自动赋予，并在 bindingTemplate 整个生命周期中有效；而 serviceKey 的值仅当 bindingTemplate 的父类容器发生变化时才会被修改。

（4）UDDI 与 WSDL。由于在 WSDL 中的服务接口表示服务的可重用定义，它在 UDDI 注册中心被作为 tModel 发布。服务实现描述服务的实例，每个实例都是使用一个 WSDL service 元素定义的。而服务实现文档中的每个 service 元素都被用于发布 UDDI businessService，即当发布一个 WSDL 服务描述时，在服务实现被作为 businessService 发布之前，必须将一个服务接口作为一个 tModel 发布，图 5-24 所示是从 WSDL 到 UDDI 的映射结构。

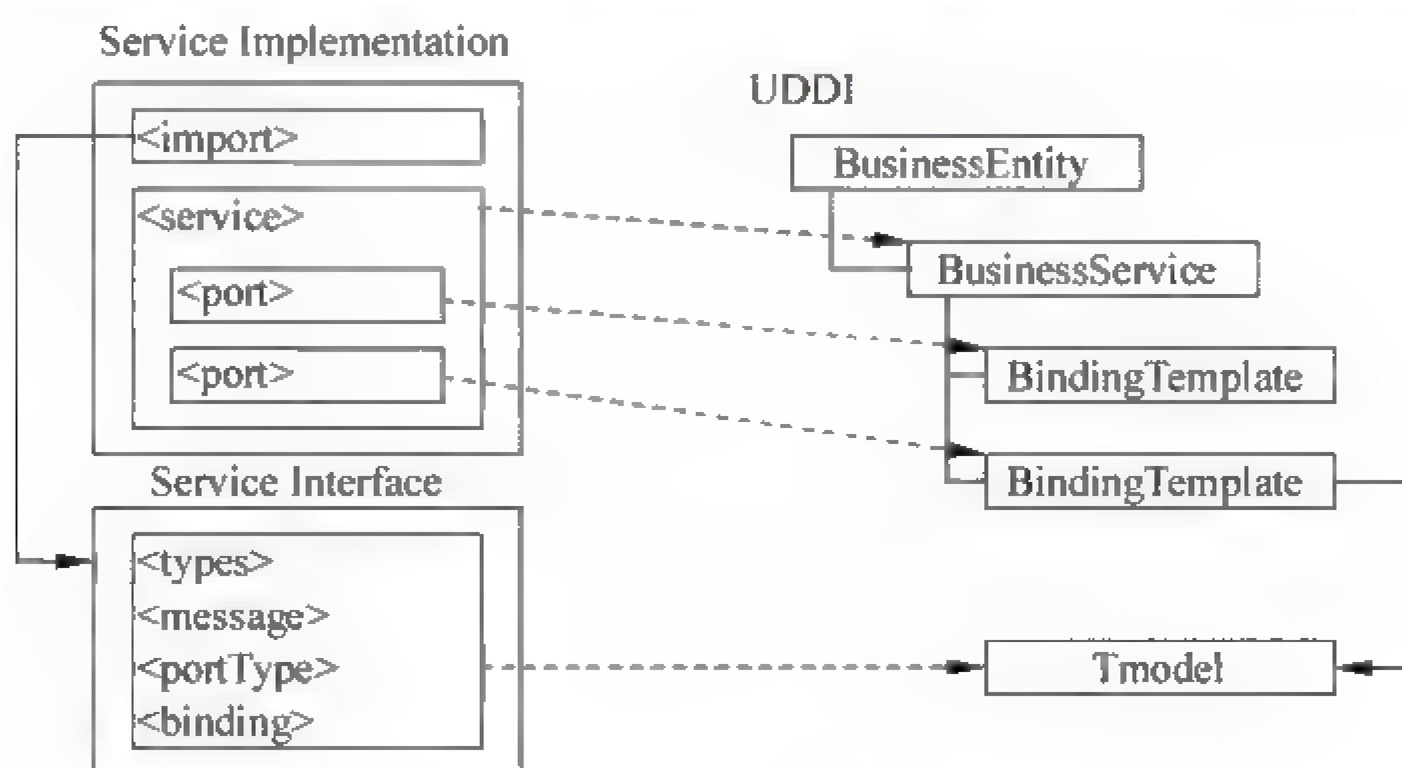

图 5-24 WSDL 到 UDDI 的映射结构

当创建 tModel 时，一个有效的 WSDL 服务接口定义的 tModel 引用应该是使用 targetNamespace 命名，并且必须包含 overviewURL（服务接口文档的位置必须在 overviewURL 元素中设置；如果服务接口文档中有多个绑定，那么必须在 URL 中对绑定进行编码）和 categoryBag（tModel 的 categoryBag 必须至少包含一个键控的引用。这个键控的引用必须包

含一个对 uddi-org:types tModel 的引用，而且键名必须是 wsdlSpec。这个条目把 tModel 当作一个 WSDL 服务接口定义)设置。如下程序片段是根据 WSDL 服务接口创建的 UDDI tModel：

```
<?xml version="1.0"?>
<tModel tModelKey="">
  <name>http://www.getquote.com/StockQuoteService-interface</name>
  <description xml:lang="en">
    Standard service interface definition for a stock quote service.
  </description>
  <overviewDoc>
    <description xml:lang="en">
      WSDL Service Interface Document
    </description>
    <overviewURL>
      http://www.getquote.com/services/
SQS-interface.wsdl#SingleSymbolBinding
    </overviewURL>
  </overviewDoc>
  <categoryBag>
    <keyedReference tModelKey="UUID:C1ACF26D-9672-4404-9D70-39B756E62AB4"
                keyName="uddi-org:types" keyValue="wsdlSpec"/>
    <keyedReference tModelKey="UUID:DB77450D-9FA8-45D4-A7BC-04411D14E384"
                keyName="Stock market trading services"
                keyValue="84121801"/>
  </categoryBag>
</tModel>
```

服务实现在 UDDI 注册中心是作为带有一个或多个 bindingTemplate 的 businessService 发布的，businessService 由服务提供者发布，如下程序片段是根据 WSDL 服务实现创建的 UDDI 商业服务。

```
<businessService businessKey="..." serviceKey="...">
  <name>StockQuoteService</name>
  <description xml:lang="en">
   Stock Quote Service
  </description>
  <bindingTemplates>
    <bindingTemplate bindingKey="..." serviceKey="...">
      <description>
        Single Symbol Stock Quote Service
      </description>
      <accesssPoint URLType="http">
       http://www.getquote.com/singlestockquote
      </accessPoint>
      <tModelInstanceDetails>
        <tModelInstanceInfo tModelKey "[tModel Key for Service Interface]">
```

```
          <instanceDetails>
            <overviewURL>
              http://www.getquote.com/services/SQS.wsdl
            </overviewURL>
          </instanceDetails>
        </tModelInstanceInfo>
      </tModelInstanceDetails>
    </bindingTemplate>
  </bindingTemplates>
  <categoryBag>
    <keyedReference tModelKey="UUID:DB77450D-9FA8-45D4-A7BC-04411D14E384"
                keyName="Stock market trading services"
                keyValue="84121801"/>
  </categoryBag>
</businessService>
```

所有的 WSDL 服务接口在 UDDI 注册中心都是作为 tModel 发布，对这些 tModel 中的每一个都要进行归类，以便将它们标识为 WSDL 服务描述。UDDI find_tModel 消息可用于查找已经分过类的 tModel。如下程序片段是查找 WSDL 服务接口描述：

```
<?xml version="1.0"?>
<find_tModel generic="1.0" xmlns="urn:uddi-org:api">
  <categoryBag>
    <keyedReference tModelKey="UUID:C1ACF26D-9672-4404-9D70-39B756E62AB4"
                keyName="uddi-org:types" keyValue="wsdlSpec"/>
  </categoryBag>
</find_tModel>
```

5．WS 常用规范

为扩展 Web 服务能力，一些新的标准已经或正在被开发。这些标准通常被冠以 WS（即 Web Service）字头。下面列举几种典型 WS 规范来进一步描述和说明。

1）WS-Security

WS-Security 是一种提供在 Web 服务上应用安全的网络传输协议。2004 年 4 月 19 日，OASIS 组织发布了 WS-Security 标准的 1.0 版本。2006 年 2 月 17 日，发布了 1.1 版本。它定义了如何在 SOAP 中使用 XML 加密或 XML 签名来保护消息传递，它并可作为 HTTPS 保护的一种替代或扩充。协议还包含了关于如何在 Web 服务消息上保证完整性和机密性的规约。同时，WS-Security 协议包括 SAML（安全断言置标语言）、Kerberos 和认证证书格式（如 X.509）的使用的详细信息，也描述了如何将签名和加密头加入 SOAP 消息。除此以外，WS-Security 还描述如何对二进制安全性权标编码。还特别描述如何对 X.509 证书和 Kerberos 票据编码，以及如何加入难于理解的加密密钥。它也还包括可以用于进一步描述消息中包含的凭证特征的扩展性机制。

（1）WS-Security 概述。WS-Security 描述通过消息完整性、消息机密性和单独消息认证提供保护质量对 SOAP 消息传递的增强，这些机制可以用于提供多种安全性模型和加密技术。同时，还提供关联安全性权标和消息的通用机制。WS-Security 不需要特定类型的安全性权标。

它在设计上就是可扩展的（例如支持多安全性权标格式）。举例来说，客户机可能会提供用户身份证明和他们有特定商业认证的证明。

通过使用 SOAP 扩展性模型，基于 SOAP 的规范被设计成与其他规范组合起来提供丰富的消息传递环境。WS-Security 自身并不保证安全性，也不提供完整的安全性解决方案。WS-Security 是一种构件，它可以与其他 Web 服务扩展和更高级的特定于应用程序的协议联合使用，以适应多种安全性模型和加密技术。但实现 WS-Security 并不意味着应用程序不会受到攻击或者安全性不会受到威胁，如下程序片段就是说明一个带有用户名安全性权标的消息。

```
<?xml version="1.0" encoding="utf-8"?>
<S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<S:Header>
<m:path xmlns:m="http://schemas.xmlsoap.org/rp/">
<m:action>http://fabrikam123.com/getQuote</m:action>
<m:to>http://fabrikam123.com/stocks</m:to>
<m:id>uuid:84b9f5d0-33fb-4a81-b02b-5b760641c1d6</m:id>
</m:path>
<wsse:Security xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext">
        wsse:UsernameToken Id="MyID">
            <wsse:Username>Zoe</wsse:Username>
        </wsse:UsernameToken>
        <ds:Signature>
           <ds:SignedInfo>
              <ds:CanonicalizationMethod Algorithm="http://www.w3.org/
              2001/10/xml-exc-c14n#"/>
             <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/
             xmldsig#hmac-sha1"/>
              <ds:Reference URI="#MsgBody">
                 <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/
                 xmldsig#sha1"/>
                 <ds:DigestValue>LyLsF0Pi4wPU...</ds:DigestValue>
              </ds:Reference>
           </ds:SignedInfo>
           <ds:SignatureValue>DJbchm5gK...</ds:SignatureValue>
           <ds:KeyInfo>
               <wsse:SecurityTokenReference>
                <wsse:Reference URI="#MyID"/>
               </wsse:SecurityTokenReference>
           </ds:KeyInfo>
        </ds:Signature>
     </wsse:Security>
  </S:Header>
  <S:Body Id "MsgBody">
```

```
        <tru:StockSymbol xmlns:tru "http://fabrikam123.com/payloads">QQQ
        </tru:StockSymbol>
      </S:Body>
    </S:Envelope>
```

（2）WS-Security 在开源软件 CXF 中的应用。Apache CXF（后续章节将详细描述 CXF）支持使用 WS-Security SOAP 扩展技术提供一整套用于消息交换的与安全相关的功能。即 CXF 使用 WS-SecurityPolicy 配置 WS-Security 安全处理。CXF 的 WS-Security 实现基于开放源码的 WSS4J（WSS4J 在 Web Services 框架中以 handler 方式工作，在发送 SOAP 消息前进行签名、加入认证凭据和加密，在收到 SOAP 消息后进行解密、认证和验证签名等安全工作）库，由 cxf-rt-ws-policy 和 cxf-rt-ws-security 模块处理。当然在 Web 服务开源软件 Axis2 也可以使用 WS-Security 来确保服务的安全，它由 Axis2 中单独发布的 Rampart 模块处理。

WS-SecurityPolicy 安全配置指定在客户机和服务之间交换的消息所需的安全处理。在大多数情况下，Web 服务堆栈还需要更多信息，才能对消息交换应用安全措施。例如，WS-SecurityPolicy 可能要求客户机对发送到服务器的请求消息进行签名，这为服务提供不可否认性。在这种情况下，在把消息发送到服务时，客户机 Web 服务堆栈需要通过某种方法确定用于签名的私有密钥。而在 CXF 中采用不同的方法，因为可以通过多种方法为 CXF 配置在对消息应用 WS-SecurityPolicy 配置时需要的参数。在客户端，可以直接在客户机代码中配置，也可以使用 Spring XML 配置文件。在服务器端，需要使用 XML 配置文件，但是仍然可以选择不同的文件类型。在实现 CXF 中实现 WS-Security 时，需要对 cxf.xml 进行静态或动态配置实现客户端服务安全，对 cxf-servlet.xml 进行动态配置确保服务端服务安全，程序片段示例分别如下，其中加黑的部分是 ws-security 的应用。

cxf.xml 动态配置：

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:jaxws="http://cxf.apache.org/jaxws"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
  http://cxf.apache.org/jaxws
  http://cxf.apache.org/schemas/jaxws.xsd">
 <jaxws:client name="{http://ws.sosnoski.com/library/wsdl}library"
     createdFromAPI="true">
     <jaxws:properties>
     <entry key="ws-security.signature.properties"
        value="client-crypto.properties"/>
     <entry key="ws-security.signature.username" value="clientkey"/>
     <entry key="ws-security.encryption.properties"
        value="client-crypto.properties"/>
     <entry key="ws-security.encryption.username" value="serverkey"/>
     <entry key="ws-security.callback-handler"
        value="com.sosnoski.ws.library.cxf.ClientCallback"/>
   </jaxws:properties>
```

```
  </jaxws:client>
</beans>
```

cxf-servlet.xml 动态配置：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:jaxws="http://cxf.apache.org/jaxws"
    xmlns:soap="http://cxf.apache.org/bindings/soap"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
      http://cxf.apache.org/jaxws
      http://cxf.apache.org/schemas/jaxws.xsd">
  <jaxws:endpoint id="Processor"
      implementor="com.sosnoski.ws.library.cxf.CXFLibraryImpl"
      wsdlLocation="WEB-INF/wsdl/library-signencr.wsdl"
      address="/">
    <jaxws:properties>
<entry key="ws-security.signature.properties" value="server-crypto.
properties"/>
      <entry key="ws-security.signature.username" value="serverkey"/>
      <entry key="ws-security.encryption.username" value="useReqSigCert"/>
      <entry key="ws-security.callback-handler"
          value="com.sosnoski.ws.library.cxf.ServerCallback"/>
    </jaxws:properties>
  </jaxws:endpoint>
</beans>
```

在使用 WS-Security 时，往往会涉及签名和（或）加密，程序清单 5-31 给出一个使用签名和加密的 WSDL 示例。

程序清单 5-31：

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://ws.sosnoski.com/library/wsdl"
    xmlns:wns="http://ws.sosnoski.com/library/wsdl"
    xmlns:tns="http://ws.sosnoski.com/library/types"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/">
...
<wsp:Policy wsu:Id="SignEncr"
      xmlns:wsu="http://docs.oasis-open.org/...-wss-wssecurity-utility-
      1.0.xsd"
      xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
      <wsp:ExactlyOne>
      <wsp:All>
```

```
<sp:AsymmetricBinding
    xmlns:sp-"http://docs.oasis open.org/ws sx/ws securitypolicy/
   200702">
  <wsp:Policy>
    <sp:InitiatorToken>
      <wsp:Policy>
        <sp:X509Token sp:IncludeToken=".../AlwaysToRecipient">
          <wsp:Policy>
            <sp:RequireThumbprintReference/>
          </wsp:Policy>
        </sp:X509Token>
      </wsp:Policy>
    </sp:InitiatorToken>
    <sp:RecipientToken>
      <wsp:Policy>
        <sp:X509Token sp:IncludeToken=".../Never">
          <wsp:Policy>
            <sp:RequireThumbprintReference/>
          </wsp:Policy>
        </sp:X509Token>
      </wsp:Policy>
    </sp:RecipientToken>
    <sp:AlgorithmSuite>
      <wsp:Policy>
        <sp:TripleDesRsa15/>
      </wsp:Policy>
    </sp:AlgorithmSuite>
    <sp:Layout>
      <wsp:Policy>
        <sp:Strict/>
      </wsp:Policy>
    </sp:Layout>
    <sp:IncludeTimestamp/>
    <sp:OnlySignEntireHeadersAndBody/>
  </wsp:Policy>
</sp:AsymmetricBinding>
<sp:SignedParts
    xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/
   200702">
  <sp:Body/>
</sp:SignedParts>
<sp:EncryptedParts
    xmlns:sp-"http://docs.oasis-open.org/ws-sx/ws-securitypolicy/
    200702">
  <sp:Body/>
```

```
        </sp:EncryptedParts>
      </wsp:All>
    </wsp:ExactlyOne>
  </wsp:Policy>
...
 <wsp:PolicyReference xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
       URI="#SignEncr"/>
...
 </wsdl:operation>
   ...
  </wsdl:binding>
  <wsdl:service name="CXFLibrary">
    <wsdl:port binding="wns:LibrarySoapBinding" name="library">
      <wsdlsoap:address location="http://localhost:8080/cxf-library-
      signencr"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

2）WS-Policy

WS-Policy 提供一个通用结构，用来配置应用到 Web 服务的特性和选项。WS-Policy 定义了一个简单 XML 结构，由四个不同元素和一对属性组成。根据 WS-Policy 解释，这些元素和属性提供一种方法来组织和合并任意复杂度的策略断言（policy assertions）。为了定义构成策略的真实断言，程序员需要使用特定扩展，比如 WS-SecurityPolicy，而不是 WS-Policy 本身。同时，为了方便起见，WS-Policy 定义了一个标准形式的策略表达式和一组可以创建更紧凑的策略表达式的规则。WS-Policy 规范也为服务请求者和服务提供者定义了语法和语义来描述他们的需求、首选项和性能，语法为以策略的形式表述每个领域的需求提供了一种灵活简洁的方法，如下程序片段是 WS-Policy 标准化形式策略：

```
<wsp:Policy>
   <wsp:All>
      <wsp:ExactlyOne>
         <nsSecurityAssertion wsp:Optional="true"/>
         <nsReliableMessagingAssertion/>
      </wsp:ExactlyOne>
      <nsTransactionAssertion/>
      <nsAuditAssertion/>
   </wsp:All>
</wsp:Policy>
```

标准形式策略表达式最多使用三个元素，但每个元素必须按照特定顺序嵌套。最外层的一般是 <wsp:Policy>，它必须且只能包含一个<wsp:ExactlyOne>子元素。而嵌套的<wsp:ExactlyOne>元素，可以包含任意多（可以为 0）的<wsp:All>子元素。因此最简单的标准形式策略表达式是<wsp:Policy><wsp:ExactlyOne/></wsp:Policy>（如上程序片段）。

标准形式策略表达式是往往冗长的，特别是如果它包含嵌套替代项。这时可以采用紧凑性策略表达式。但一个策略的多个紧凑表达式等价于一个标准表达式。紧凑表达式选项的一个特性是，拥有通过基本策略元素（在策略术语中称为操作，因为每个元素暗含嵌套断言的一个具体解释）任意次序嵌套来表达策略的能力。嵌套规则也定义一种直接使用的<wsp:Policy>元素（不需要标准形式中必须有的<wsp:ExactlyOne>子元素）的解释等价于一个<wsp:All>，这可能是最广泛使用的紧凑表达式特性，如下程序片段是紧凑形式策略表达式：

```
<wsp:Policy>
  <sp:AsymmetricBinding
      xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
    <wsp:Policy>
      <sp:InitiatorToken>
        <wsp:Policy>
          <sp:X509Token
              sp:IncludeToken=".../IncludeToken/AlwaysToRecipient">
            <wsp:Policy>
              <sp:RequireThumbprintReference/>
            </wsp:Policy>
          </sp:X509Token>

        </wsp:Policy>
      </sp:InitiatorToken>
      ...
```

除了简化元素嵌入，紧凑表达式也提供一种引用和重用策略表达式的方法。程序员可以使用第四个 WS-Policy 元素<wsp:PolicyReference> 来实现。<wsp:PolicyReference>元素可以在任何策略断言出现的地方出现。引用的策略表达式被策略引用（Policy reference）有效替代（技术上是使用<wsp:All>元素替代引用的 <wsp:Policy> 元素），程序片段如下：

```
<wsp:Policy wsu:Id="ClientX509"
    xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd">
  <sp:InitiatorToken>
    <wsp:Policy>
      <sp:X509Token
          sp:IncludeToken=".../IncludeToken/AlwaysToRecipient">
        <wsp:Policy>
          <sp:RequireThumbprintReference/>
        </wsp:Policy>
      </sp:X509Token>
    </wsp:Policy>
  </sp:InitiatorToken>
</wsp:Policy>
<wsp:Policy>
  <sp:AsymmetricBinding
      xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
```

```
    <wsp:Policy>
      <wsp:PolicyReference URI="#ClientX509"/>
      ...
```

在前述章节的 WSDL 概述和分析中，已经明白在 WSDL1.1 中服务定义使用了层次结构，第一层（底层）由<wsdl:message>元素构成，定义消息的 XML 结构到服务，或者从服务定义。第二层是<wsdl:portType>元素，定义操作集，每一个操作由输入、输出或默认消息指定。第三层是<wsdl:binding>元素，使用<wsdl:portType>将一个特定消息协议和访问方法联系到一起。第四层<wsdl:portType>元素格式的服务器端定义，这指定<wsdl:binding>的访问路径。WS-Policy 允许向 WSDL 服务定义附加策略，但这几个点不能精确匹配定义的层。然而，层代表的是服务定义的一个逻辑结构，WS-Policy 更关注消息和消息分组。WS-Policy 使用的四个消息分组级别是：

（1）消息：策略适用于一个特定消息。如果策略是通过<wsdl:message>元素附加的，消息可用于任何地方；如果在<wsdl:portType>或<wsdl:binding>元素中策略是通过操作的 input/output/fault 定义附加的，消息可以被一个特定操作所用。

（2）操作：策略适用于一个特定操作的所有消息交换（在<wsdl:binding>或<wsdl:portType>中，通过<wsdl:operation>元素附加的策略）。

（3）端点：策略适用于一个特定服务绑定的所有消息交换（通过<wsdl:port>或<wsdl:binding>附加的策略），或者适用于基于一个特定端口类型的所有服务绑定的消息交换（附加到<wsdl:portType>的策略）。

（4）服务：策略适用于所有端点和所有与服务关联的操作（在<wsdl:service>元素附加的策略）。

程序片段是策略附加示例如下。

```
......
<wsdl:binding name="LibrarySoapBinding" type="wns:Library">
  <wsp:PolicyReference xmlns:wsp="http://www.w3.org/ns/ws-policy" URI=
  "#UsernameToken"/>
  <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/
  soap/http"/>
  <wsdl:operation name="addBook">
    <wsp:PolicyReference xmlns:wsp="http://www.w3.org/ns/ws-policy" URI=
    "#AsymmEncr"/>
    ...
  </wsdl:operation>
</wsdl:binding>
...
```

3）WS-Addressing

定义了在 SOAP 消息内描述发送/接收方地址的方式。WS-Addressing 提供一种方式来指定关于位置的信息，而不只是一个统一资源标识符（Universal Resource Identifier，URI 或 URL）。WS-Addressing 实际上只包括两个新概念：端点引用（endpoint reference，EPR）和

SOAP 结构的消息信息（message information，MI），其中 EPR 包括目的地（该消息的目的地的 URI），源端点（发出该消息的服务端点），应答端点（应答消息接收者的端点），故障端点（故障消息接收者的端点），动作[指示该消息的语义（可能有助于该消息的寻址）的 URI]，消息 ID（唯一消息标识符 URI），关系（与之前消息的关系）。

如下程序片段分别是 EPR 和 MI 的表达方式：

EPR 表达方式程序片段：

```
<wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/02/addressing"
xmlns:sat="http://example.org/satelliteSystem">
   <wsa:Address>http://example.com/satellite</wsa:Address>
   <wsa:ReferenceProperties>
       <sat:SatelliteId>SAT9928</sat:SatelliteId>
   </wsa:ReferenceProperties>
</wsa:EndpointReference>
```

MI 表达方式程序示例片段（Web 服务的任何响应都应该被发送给 ReplyTo 端点引用）：

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:f123="http://www.fabrikam123.example/svc53">
  <S:Header>
  <wsa:MessageID>uuid:aaaabbbb-cccc-dddd-eeee-ffffffffffff
    </wsa:MessageID>
    <wsa:ReplyTo>
    <wsa:Address>http://business456.example/client1</wsa:Address>
    </wsa:ReplyTo>
    <wsa:To S:mustUnderstand="1">mailto:joe@fabrikam123.example</wsa:To>
    <wsa:Action>http://fabrikam123.example/mail/Delete</wsa:Action>
  </S:Header>
  <S:Body>
    <f123:Delete>
      <maxCount>42</maxCount>
    </f123:Delete>
  </S:Body>
</S:Envelope>
```

5.6.2　语义 Web 服务技术

Web 服务是基于 XML 实现的，因此，Web 服务的操作是由 XML 来完成。由于 XML 只有深层次的语法识别，没有语义级的检查和验证，也就给 Web 服务智能、自动的完成业务逻辑带来了挑战。而语义 Web 于 2001 年由 Berners-Lee、Hendler 和 Lassila 最初提出；语义 Web 背后的关键技术是资源描述框架（RDF），它既可被视为图形（N-三元组图表）知识表示模型，也可被视为面向对象的知识表示模型。要让这个模型成为机器可读的模型，有若干种格式可用（比如 RDF/XML、RDF/N3 和 RDF/Turtle），从而将万维网中现存的信息发展成一个巨大

的全球信息库、知识库。它研究的主要目的就是扩展当前的万维网，使得网络中的信息具有语义，能够被计算机理解，便于人和计算机之间的交互与合作，其研究重点就是如何把信息表示为计算机能够理解和处理的形式，即带有语义。TimBerners-Lee 给出了语义 Web 中的层次结构关系，它主要基于 XML 和 RDF/RDFS，并在此之上构建本体（Ontology）和逻辑推理规则，以完成基于语义的知识表示和推理，从而能够为计算机所理解和处理。而语义 Web 服务是语义 Web 与 Web 服务结合产物，它主要方法是利用本体来描述 Web 服务，然后通过这些带有语义信息的描述 Web 服务来实现服务的自动发现，选择和组合。也使得语义 Web 和 Web 服务是语义 Web 服务的两大支撑技术。目前描述语义 Web 服务的语言是 OWL-S，它是连接两大技术的桥梁，而对语义 Web 服务置标语言研究最典型的组织就是 DARPA 组织，其研究组 OWL Services Coalition 提出了语义 Web 服务置标语言就是 OWL-S。相关语义 Web 服务技术在第 3 章进行了详细概述、分析和总结。

5.6.3 RESTful Web 服务技术

REST（Representational State Transfer）是 Roy Fielding 博士在 2000 年他的博士论文中提出来的一种软件架构风格。因为 REST 模式的 Web 服务与复杂的 SOAP 和 XML-RPC 对比来讲，它是非常明显的且更加简洁。目前，越来越多的 Web 服务开始采用 REST 风格设计和实现不同的软件系统。而 RESTful Web 服务（又称 RESTful Web API）是一个使用 HTTP 并遵循 REST 原则的 Web 服务。例如，Amazon.com 提供接近 REST 风格的 Web 服务进行图书查找；雅虎提供的 Web 服务也是 REST 风格的。REST 通常基于使用 HTTP，URI，和 XML，以及 HTML 这些现有的广泛流行的协议和标准。

REST 定义了一组体系架构原则，程序员可以根据这些原则设计以系统资源为中心的 Web 服务，包括使用不同语言编写的客户端如何通过 HTTP 处理和传输资源状态。如果考虑使用它的 Web 服务的数量，REST 近年来已经成为最主要的 Web 服务设计模型。事实上，REST 对 Web 的影响非常大，由于它使用相当方便，在一定的领域里，已经普遍地取代了基于 SOAP 和 WSDL 的接口设计。这是因为 RESTful Web 服务使用标准的 HTTP 方法（GET/PUT/POST/DELETE）来抽象所有 Web 系统的服务能力，而 SOAP 应用都通过定义自己个性化的接口方法来抽象 Web 服务。RESTful Web 服务使用标准的 HTTP 方法优势，其标准化的 HTTP 操作方法与 URI，HTML，XML 等标准技术结合，将会极大提高系统与系统之间整合的互操作能力，尤其在 Web 应用领域，RESTful Web 服务所表达的这种抽象能力更加贴近 Web 本身的工作方式，也更加自然和方便。图 5-25 所示是一种 REST 实现架构实现。

基于 REST 的 Web 服务的主要特征之一是以遵循 RFC 2616 定义的协议的方式来显式使用 HTTP 方法。例如，HTTP GET 被称作数据产生方法，旨在由客户端应用程序用于检索资源以从 Web 服务器获取数据，或者执行某个查询并预期 Web 服务器将查找某一组匹配资源，然后使用该资源进行响应。从而使得 REST 要求开发人员显式地使用 HTTP 方法，并且使用方式与协议定义一致。这个基本 REST 设计原则建立了创建、读取、更新和删除（create, read, update, and delete，CRUD）操作与 HTTP 方法之间的一对一映射。

REST Web 服务使得具有负载平衡和故障转移功能、代理和网关的服务器集群通常以形成服务拓扑的方式进行组织，从而允许根据需要将请求从一个服务器路由到另一个服务器，

以减少 Web 服务调用的总体响应时间。要使用中间服务器扩大规模，REST Web 服务需要发送完整、独立的请求；也就是说，发送的请求包括所有需要满足的数据，以便中间服务器中的组件能够进行转发、路由和负载平衡，而不需要服务请求间在本地保存任何状态。并且完整、独立的请求不要求服务器在处理请求时检索任何类型的应用程序上下文或状态。REST Web 服务应用程序（或客户端）在 HTTP Header 和请求正文中包括服务器端组件生成响应所需要的所有参数、上下文和数据。

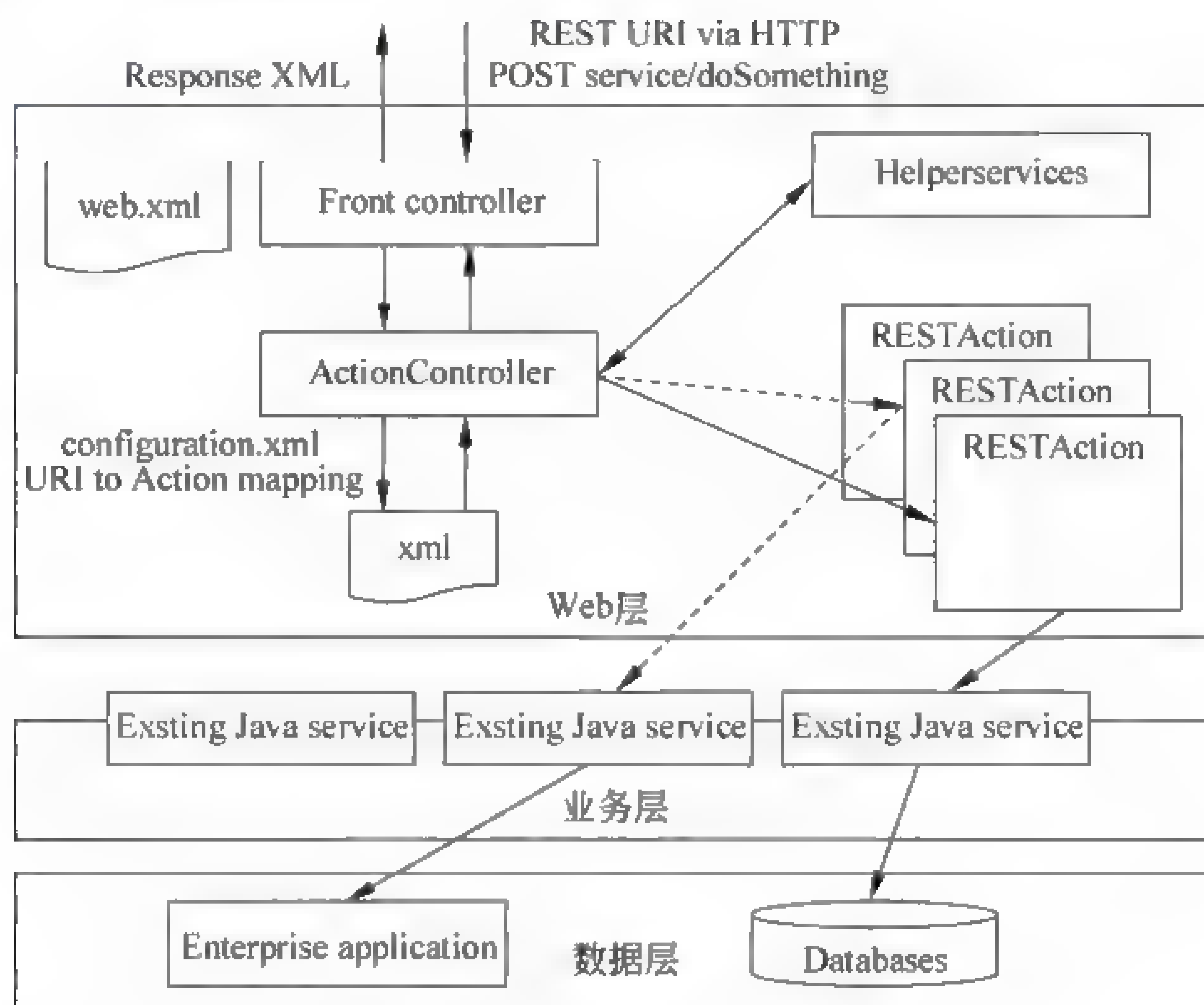

图 5-25　一种 REST 架构实现

在 RESTWeb 服务实现方面，由于具有轻量级特性，以及通过 HTTP 直接传输数据的特性，Web 服务的 RESTful 方法已经成为最常见的替代方法，使得可以使用各种语言（比如 Java 程序、Perl、Ruby、Python、PHP 和 Javascript+Ajax）来实现客户端。RESTful Web 服务通常可以通过自动客户端或代表用户的应用程序访问。但是，这种服务的简便性让用户能够与之直接交互，使用它们的 Web 浏览器构建一个 GET URL 并读取返回的内容。这时，erome Louvel 和 Dave Pawson 开发的 Restlet 是轻量级的 REST Web 服务实现框架，它实现针对各种 RESTful 系统的资源、表示、连接器和媒体类型之类的概念，包括 Web 服务。在 Restlet [①]框架中，客户端和服务器都是组件，并且组件通过连接器互相通信。该框架最重要的类是抽象类 Uniform 及其具体的子类 Restlet，该类的子类是专用类，比如 Application、Filter、Finder、Router 和 Route。这些子类能够一起处理验证、过滤、安全、数据转换，以及将传入请求路由到相应资源等操作。而 Resource 类生成客户端的表示形式。同时，JSR-311 是 Sun Microsystems 的规范，也可以用于开发 RESTful Web 服务，并能定义一组 Java API；其中 Jersey 框架是对 JSR-311 的参考实现，它所提供一组注释、相关类和接口都可以用来将 Java 对象作为 Web 资源展示，该框架假定 HTTP 是底层网络协议。它使用注释提供 URI 和相应资源类

① http://www.restlet.org/

之间的清晰映射，以及 HTTP 方法与 Java 对象方法之间的映射。且 API 支持广泛的 HTTP 实体内容类型，包括 HTML、XML、JSON、GIF、JPG 等。它还将提供所需的插件功能，以允许使用标准方法通过应用程序添加其他类型。而 JAX-RS 与 RESTlet API 的不同之处在于，在 RESTlet 下，REST 资源是结构化组织起来的，如 Component 可以包含多个 Application，Application 又可以包含多个 REST 资源，Component 到 Application，Application 到 REST 资源用 Route 来连接。这样，从 URI 到 REST 资源的定位就自上而下进行查找。而 JSR-311 下，REST 资源是 POJO 并且非结构化的，资源对应的 URI 通过 Annotation 直接在 POJO 类里加以描述（这里不讨论 subresource 资源的定位）。相对来说，JAX-RS 描述能力简单，开发起来更加方便。

JAX-RS Extension 是在 RESTlet 架构下的对 JAX-RS:Java API for RESTful Web Services 的实现，它实现了 JAX-RS。主要的技术要点包括：

（1）internal.Provider：JAX-RS Extension 通过 MessageBodyReader 和 MessageBody-Writer 实现了 InputStream、Jaxb、ByteArray 等 Provider 的序列化和反序列化。

（2）internal.spi：JAX-RS Extension 实现了 HTTP 协议基础类。

（3）internal.exceptions：定义了 JAX-RS 抛出的异常。

（4）JaxRsApplication：实现了 JAX-RS 的 Application 接口，并且包含 JaxRsRestlet，但具体的工作是在 JaxRsRestlet 中处理的。

（5）JaxRsRestlet：包含了所有的 REST POJO 资源，在初始化时分析 REST 资源的 Annotation，得到 REST 资源所对应的 URI，对应的接口及其他相关信息，JaxRsRestlet 还包含了所有 Provider 的引用和所有异常的引用。在运行过程中，所有的 REST 请求被路由到 JaxRsRestlet，并由该对象来选择合适的 REST 资源及方法来进行处理。下列一段程序代码是基于 RESTlet Jax-Rs Extension 的 Web Service 部署方法。它与部署一个基本的 Servelet 极其相似。不同的是，部署过程中，用户需要注意添加需要的 Jar 包。然后，创建 Servelet 的配置文件 web.xml。下面是为本例所写的配置文件（最后用户需要将这些一起打包成 WAR 包，并部署到用户选定的 Servelet 容器中完成部署）：

```
<?xml version="1.0" encoding="UTF-8"?>
 <web-app id="WebApp ID" version="2.4"
          xmlns="http://java.sun.com/xml/ns/j2ee"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
              http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
   <display-name>RESTlet Jax-RS extension Example</display-name>
   <!-- Application class name -->
   <context-param>
      <param-name>org.restlet.application</param-name>
      <param-value>
         com.developerworks.jaxrs.resltet.example.JaxRsExtensionApplication
      </param-value>
   </context-param>
  <!-- Restlet adapter -->
   <servlet>
```

```
      <servlet name>RestletServlet</servlet name>
      <servlet class>
        com.noelios.restlet.ext.servlet.ServerServlet
      </servlet-class>
    </servlet>
   <!-- Catch all requests -->
    <servlet-mapping>
      <servlet-name>RestletJaxRsExtensionServlet</servlet-name>
      <url-pattern>/*</url-pattern>
    </servlet-mapping>
  </web-app>
```

5.6.4　SOA 技术

如果只是将 Web 服务作为通信协议就不能实现真正的面向服务的体系结构（SOA），如图 5-26 所示。SOA 描述了服务的整个系统如何动态地相互查找，如何聚集在一起执行某些应用程序，以及如何按照多种方式进行组合。同时，SOA 鼓励技术和软件的重用，从而发展了设计、开发和使用应用程序的方式，并且它所使分布式计算更接近于现实。但在这一层次上，软件开发人员需要考虑 SOA 模型并通过此模型来设计他们的分布式应用程序。这一层的特点是使用各种技术来支持服务的分布式计算，比如企业服务总线（ESB），它是一个通用的分布网络，可以与服务一起协同工作。而最高的层次的应用是把 SOA 模型和许多组件服务看作是构件，这些构件可以装配成作为一个整体的某些部分并放入完整的应用程序中，而不是用传统的方法来编写一行一行的代码。通过检验连接接口，程序员就可以不真正编写一行代码的情况下构建整个应用程序。

事实上，按照这种方式，甚至还可能得到直接代码，因为服务可以通过多种不同的语言和平台进行编写。并且可以将构件放在一起来组成应用程序执行方式的操作工作流，而且还可以用其他工具来监控每个服务或服务组的工作流的有效性。在这一层次上，开发人员还可以把常规编程语言放在一边，并使用模型驱动的体系结构（Model-Driven Architecture）来帮助他们构建设计更精确的应用程序。然后，这样设计的应用程序就可以运行在分布式系统（如 ESB，相关 ESB 详细描述在前面章节已经详细描述了）之上。

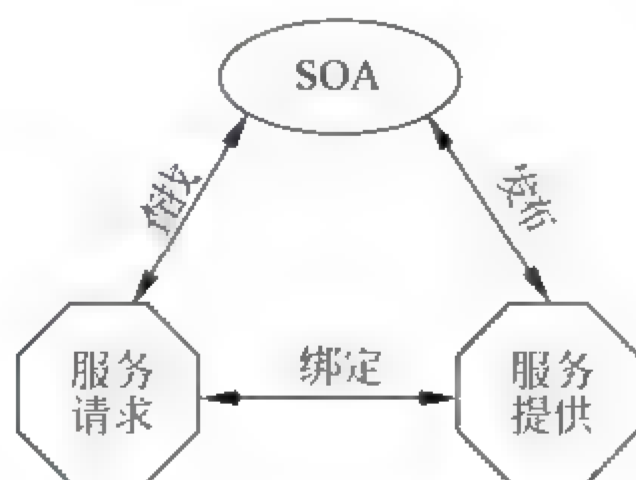

图 5-26　基于 Web 服务的 SOA 结构

1. SCA

SCA（Service Component Architecture）即服务组件体系结构，它提供了一个编程模型来

构建和开发基于 SOA 的应用系统。SCA 于 2005 年 11 月 30 日，由 IBM、BEA、Oracle、SAP、Iona、Siebel 和 Sybase 等公司正式推出，版本号为 0.9。2007 年，IBM、BEA 等 SCA 规范参与厂商共同成立 OSOA(Open Service Oriented Architecture)组织来制定和推动 SCA 以及 SDO（Service Data Objects）规范，同年 3 月，OSOA 组织正式发布 SCA 系列规范 1.0 版本。表 5-6 所示是 0.9 版与 1.0 版所提供的功能比较。

表 5-6　SCA 0.9 版与 1.0 版功能比较

	SCA 0.9	SCA 1.0
组件层次	不支持模块作为组件的实现	支持 Composite 作为组件的实现
模块层次	External Service	Reference
	Entry Point	Service
	无此概念，通过 context 调用服务	无此概念，通过 context 调用服务
	Module	Composite
	无嵌套装配模型	支持嵌套装配
	Module 不支持 property 配置	Composite 支持 property 配置
打包和部署层次	subsystem	Domain
	Module Fragment	Composite include
	支持 Module 和 subsystem 的打包和部署	支持 Composite 和 Contribution 部署到 Domain 中
QoS	初略定义了 QoS 框架	详细的 QoS 框架（由策略框架规范定义）
其他	有扩展模型，支持扩展接口、实现和绑定	有扩展模型，支持扩展接口、实现和绑定
	支持 Java Annotation	支持 Java Annotation

1）SCA 概述

SCA 是一组规范，如图 5-27 所示。UML 结构如图 5-28 和图 5-29 所示。SCA 描述了用于使用 SOA 来构建应用程序和系统的模型，它扩展了以前用于实现服务的方法，并对其形成补充，而且，SCA 构建于 Web 服务系列标准等开放标准之上，从而使得要在模型结构上了实现了 SOA。在目前 SCA 0.9 规范中，SCA 支持以下特性，它们之间关系如图 5-27 所示。

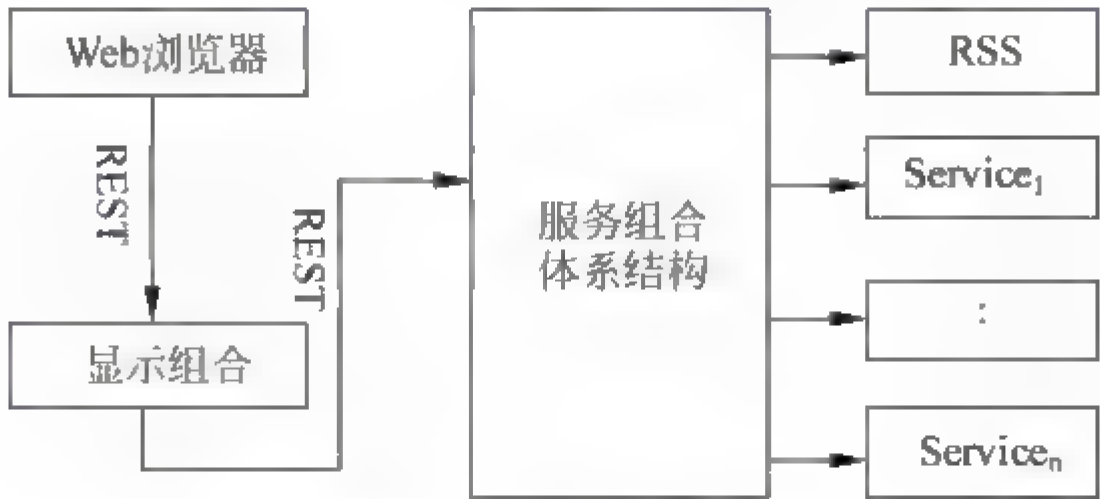

图 5-27　SCA 关系图

SCA 支持使用众多编程语言中的任何一种语言编写的服务实现，这既包括常规的面向对象和过程的语言，如 Java、PHP、C++、COBOL，也包括 BPEL 和 XSLT 等以 XML 为中心的语言，还包括 SQL 和 XQuery 等面向问题的语言。SCA 还支持各种编程样式，除了同步调

用——返回样式外，还包括异步样式和面向消息的样式。

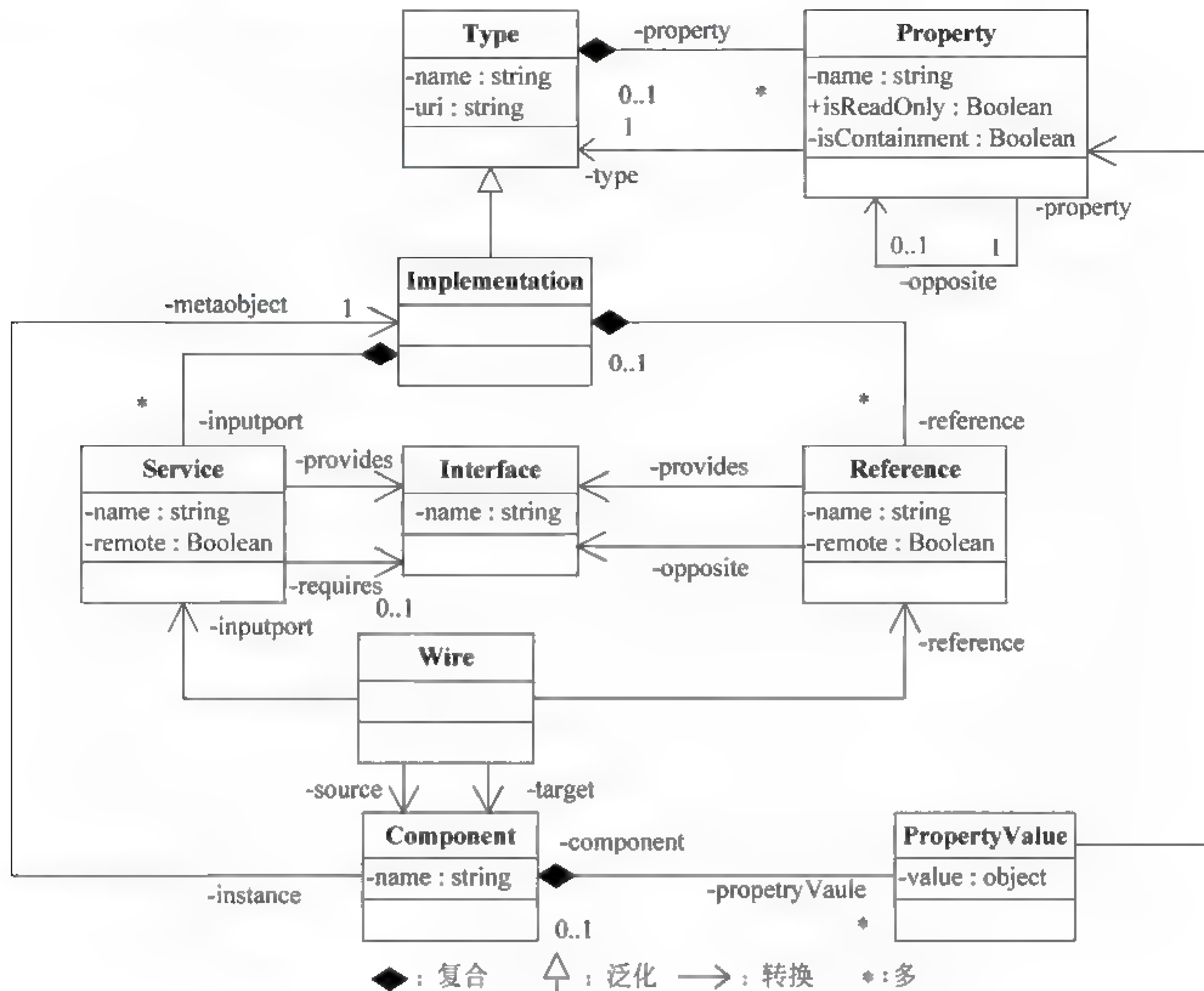

图 5-28　SCA 的 UML 表示（1）（来自 SCA 规范）

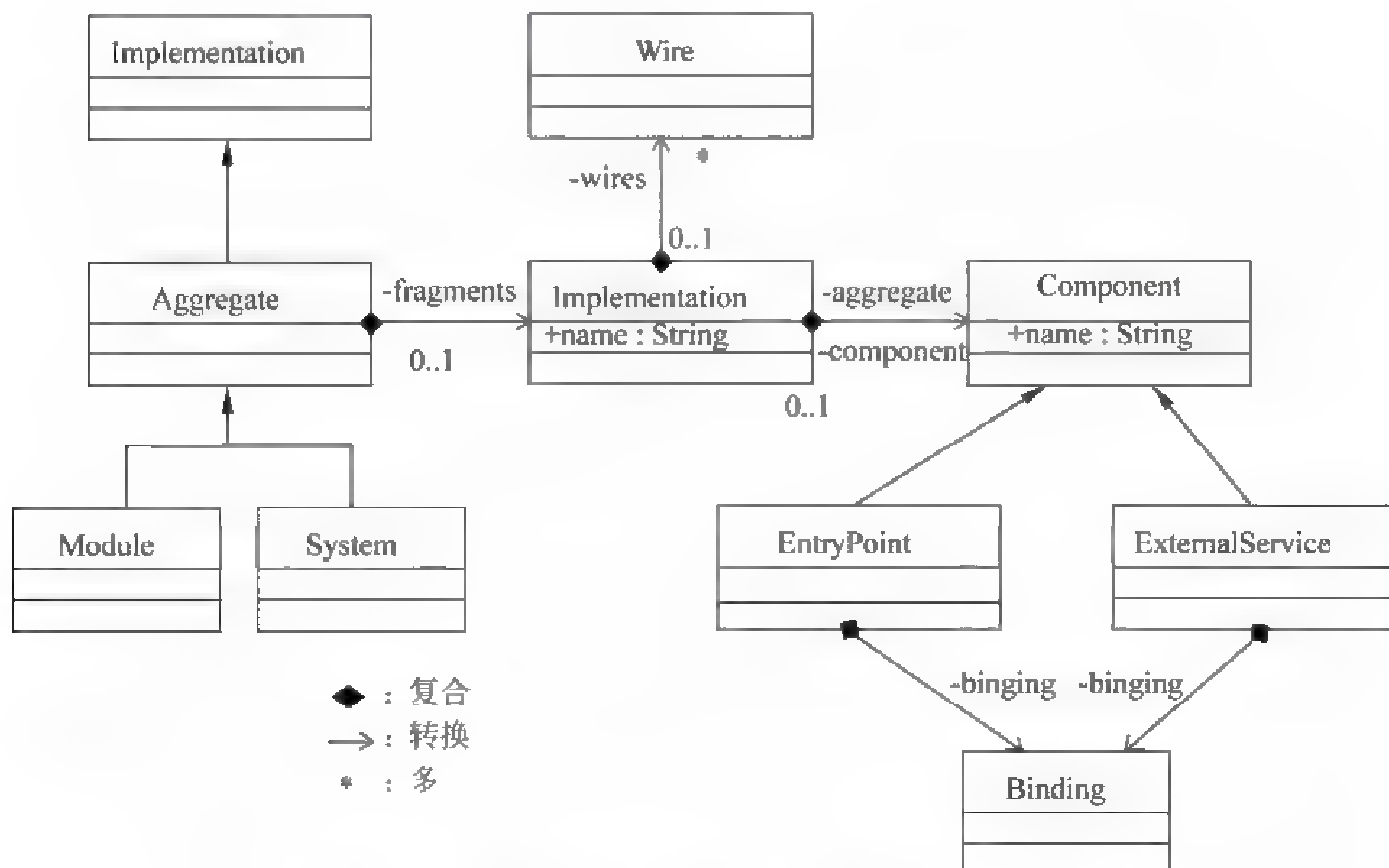

图 5-29　SCA 的 UML 表示（2）（来自 SCA 规范）

SCA 支持到各种用于调用服务的访问机制的绑定，这包括 Web 服务、消息传递系统和 CORBA IIOP。绑定是以声明方式处理的，独立于实现代码。而基础设施功能（如安全、事务和可靠消息传递的使用）都是采用声明方式来实现代码分离。SCA 通过策略使用基础设施功能，而策略旨在简化控制功能如何应用到业务系统的机制。

SCA 还促进了使用服务数据对象（SDO）来表示形成参数和服务返回值的业务数据，从而提供了对业务数据的统一访问方式，从而对 SCA 本身提供的业务服务统一访问方式形成了补充。

SCA 规范分为多个文档，每个文档分别处理 SCA 的不同方面。组装模型（Assembly Model）通过连接处理组件的链接关系，它是独立于实现语言的。客户机和实现（Client and Implementation）规范处理服务和服务客户机的实现——每种语言都有自己的客户机和实现规范，如表 5-7 所示，用于描述该语言的 SCA 模型。

表 5-7　SCA 规范

规范分类	规范名称
核心规范	SCA Assembly Model
	SCA Policy Framework
	SCA Transaction Policy
	SCA Assembly Extensions for Event Processing
组件实现技术规范	SCA Java Common Annotations and APIs
	SCA Java Component Implementation
	SCA Spring Component Implementation
	SCA BPEL Client and Implementation
	SCA C++ Client and Implementation
	SCA COBOL Client and Implementation
	SCA C Client and Implementation
	SCA Java EE Integration Specification
绑定技术规范	SCA Web Services Binding
	SCA JMS Binding
	SCA EJB Session Bean Binding
	SCA JCA Binding

2）SCA 基本应用描述

SCA 的目的是成为构建 SOA 应用的编程模型，它的应用领域是企业应用集成领域。SOA 虽然提了很多年，但到目前，在概念方面，各大厂商关于 SOA 也都有着不同的理解；在实现技术方面，也大都是在使用具体的 Web Service，但 Web Service 是一种远程调用技术且是一种基于 XML 的技术，对于如何开发本地 SOA 应用中的服务，目前也无统一认知。在组件模型方面，目前也没有统一的组件模型来装配和管理服务。所以，当前的 SOA 应用开发缺乏服务组件模型支持，缺乏从组件层次上来进行服务声明、发布、组装及定义互操作性。另外，基于企业应用的特点，不同的应用系统可能有不同的实现技术。如何能将这些采用不同的技术来实现的功能发布为 SOA 中的服务，并能够更多的关注服务声明，而屏蔽底层的实现，也是 SOA 应用开发中需要关注的一个问题。而 SCA 通过一系列规范的定义（见 5.6.4.1 小节），

有效解决了上述问题，即组件装配模型提供了对服务声明、发布、组装，并通过绑定功能，来屏蔽不同服务间的通信细节；支持不同的编程技术作为服务组件的实现，并可以更好的将遗留系统封装为系统中的服务；策略框架同时提供了对应用系统中 QoS 的支持，以保证企业应用中的质量需求。图 5-30 是 SCA 在企业级应用的体系结构示例图。

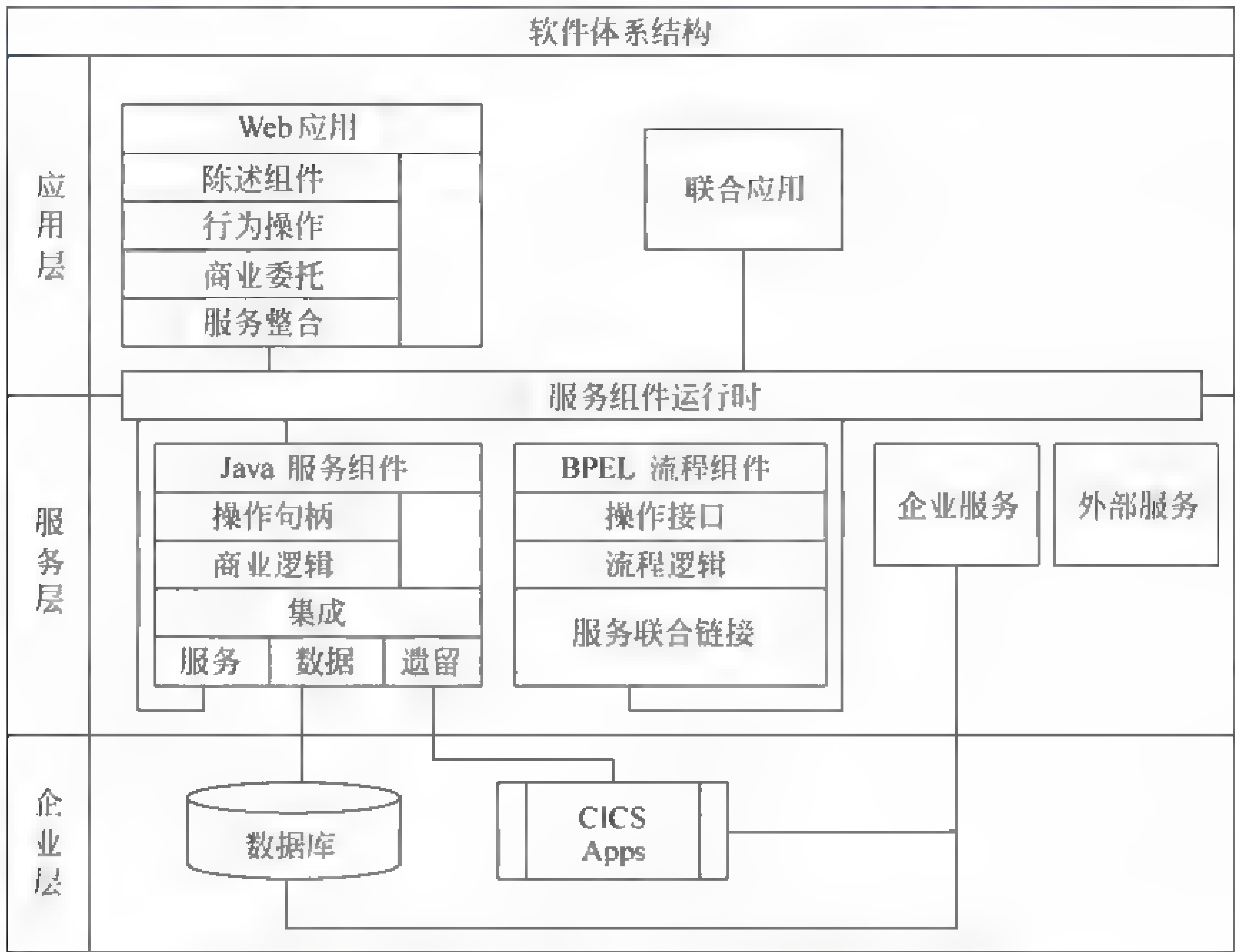

图 5-30　面向 SOA 的 SCA 企业级应用的体系结构

SCA 服务装配模型是整个 SCA 规范中的核心，并在设计过程就偏重于 SCA 组装；且定义了灵活的组件装配模型，特别是提供了可嵌套的组件装配模型，可以由最小的原子组件组装成一个大系统。在 SCA 服务绑定方面，SCA 的服务绑定需要从两个层次上看，在同一个复合组件内部中的引用，如果没有设定自动连线，那它实际上采用的设计期绑定，因为在设计期就需要指定所需要引用的组件及其服务，这是因为 SCA 不支持组件和服务的版本管理，所以在运行期永远只有一个同名组件存在，所以也就没必要支持运行期绑定；在跨复合组件和跨域间的引用，SCA 采用的是运行期绑定，因为复合组件是 SCA 的最小可部署单元，开发人员可能会在运行期重新部署同名的复合组件。

同时，SCA 定义了一种 XML 语言服务组件定义语言（Service Component Definition Language，SCDL），该语言允许开发人员定义组件并将其连接起来，例如程序清单 5-32：

程序清单 5-32　(SCDL 实现基于 SCA 组件的组装实例代码)：

```
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0" name="sample.alerter">
  <service name="AlerterService">
    <binding.rest/>
    <reference>参考的组件名称</reference>
  </service>
  <component name="参考的组件名称">
```

```
        <implementation.python module="组件实现(含组件实现的语言，这里用 python 语言)
        " scope="composite"/>
        <reference name="组件名称 1">组件名称 1</reference>
        <reference name="组件名称 2">组件名称 2</reference>
        <reference name="组件名称 3">组件名称 3</reference>
        ...
      </component>
      <component name="组件名称 1">
        <implementation.python module="组件名称 1 实现" scope="composite"/>
      </component>
      <component name="组件名称 2">
        <implementation.python module="组件名称 2 实现" scope="composite"/>
      </component>
      <component name="组件名称 3">
        <implementation.python module="组件名称 3 实现" scope="composite"/>
      ...
      </component>
    </composite>
```

开发一个基于 SOA 应用系统，编程模型和数据模型是其两个重要方面。SCA 用于提供编程模型，SDO 则提供了数据模型（在 5.6.4.2 小节进行描述），SDO 致力于为应用系统中处理数据提供统一的方式。而在一个大型 SOA 应用中，一个常见的问题便是不同的服务间需要通过不同的访问协议来进行互操作，绑定技术规范便是用于定义 SCA 服务或引用所支持的访问协议，SCA 支持多种绑定技术，其可扩展框架同时支持开发人员添加其他绑定技术，目前 SCA 定义了 JMS、Web Service 等绑定技术，程序清单代码 5-33 就是实现 Web Service 绑定技术方法（该代码来自 SCA 0.9 规范）。

程序清单 5-33：

```
<?xml version="1.0" encoding="ASCII"?>
  <module xmlns="http://www.osoa.org/xmlns/sca/0.9"
xmlns:v="http://www.osoa.org/xmlns/sca/values/0.9" name="MyValueModule" >
<entryPoint name="MyValueService">
<interface.java interface="services.myvalue.MyValueService"/>
<binding.ws port="http://www.myvalue.org/MyValueService#
wsdl.endpoint(MyValueService/MyValueServiceSOAP)"/>
...
</entryPoint>
...
<externalService name="* ">
<interface.java interface="* "/>
<binding.ws port="http://www.*.org/StockQuoteService#
wsdl.endpoint()"/>
</externalService>
</module>
```

2. SDO

正如前面所述，SCA 为实现 IT 服务提供了一个开放的、与技术无关的模型，它根据业务功能定义服务，使应用程序开发人员更易于访问中间件功能。SCA 还为单个服务集合的业务解决方案的组装提供了一个模型，并控制解决方案的各个方面，如访问方法和安全。而 SDO 提供了访问许多不同种类数据的公共方法，从而对 SCA 进行了补充。该规范可以减少访问和操作业务数据所需的技能级别和时间。现在，大多数 API 都可以用于操作数据。这些 API 往往紧密耦合源数据和目标数据，因而在使用时易于出错，并且在业务需求发展时易于中断。而 SDO 使得利用这些 API 和实现它们的价值更加容易，并且不需要直接对其进行编码。参与制定 SDO 的供应商包括 BEA Systems、IBM、Oracle、SAP、Siebel、Sybase 和 Xcalia，图 5-31 所示是 SDO 结构。

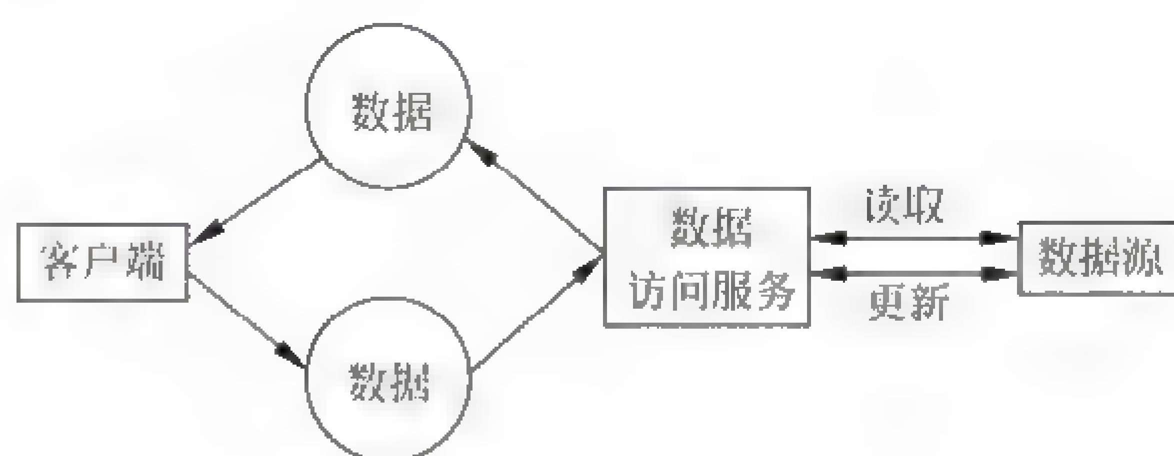

图 5-31　SDO 结构

1）SDO 概述

2005 年 6 月发布了 Java SDO Specification 的更新扩展版本——Version 2.0。Java SDO Specification 现在可与 C++ SDO Specification 进行互补。二者均已发布了 Version 2.01。这表示仅对 Java 规范进行了很小的编辑性更改，但这是 SDO 规范的第一个公开版本。SDO 设计用于简化和统一应用程序处理数据的方式。通过使用 SDO，应用程序编程人员可以采用统一的方式访问和操作来自异类数据源的数据，包括关系数据库、XML 数据源、Web 服务以及企业信息系统中数据等。SDO 基于断开连接的数据图的概念，数据图是树形结构或图形结构的数据对象的集合。在断开连接的数据图形体系结构下，客户机将从数据源检索数据图，对数据图进行变异操作，然后将数据图更改应用回数据源。图 5-32 所示是 SDO 数据图结构。图 5-33 所示是 SDO 数据图 API 结构。

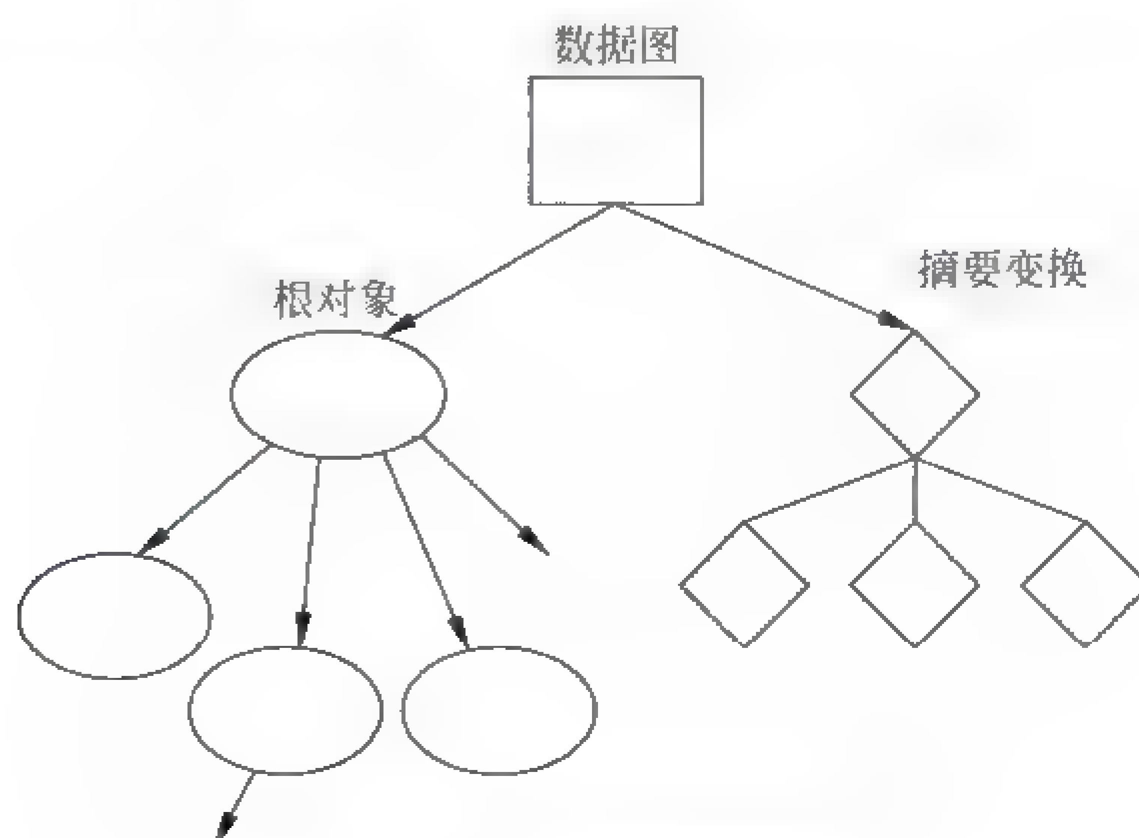

图 5-32　SDO 数据图

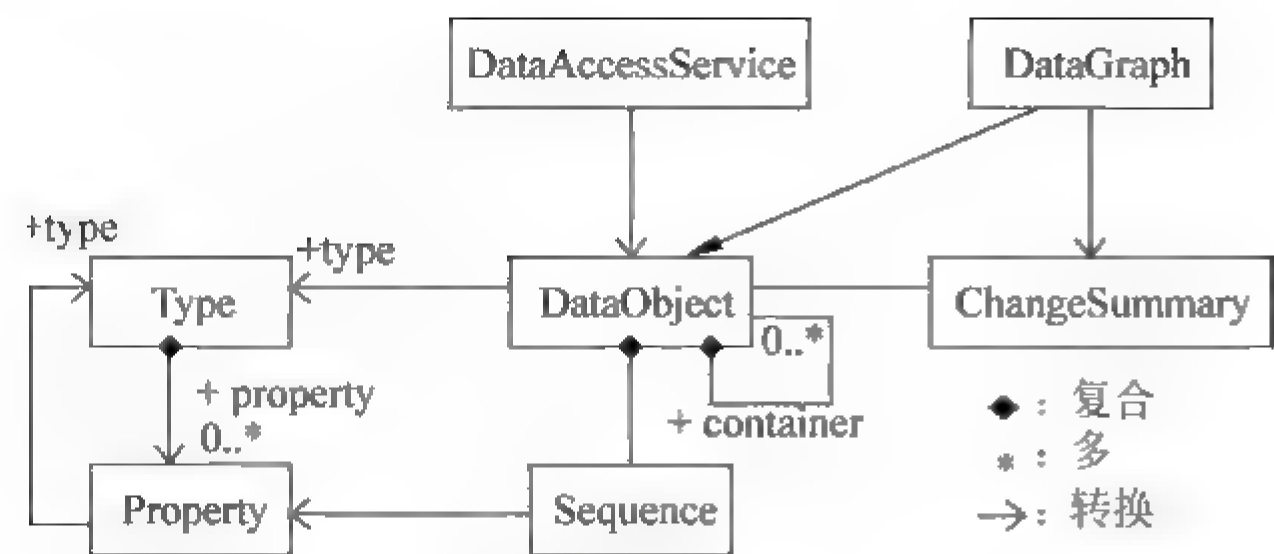

图 5-33　SDO API

而 SDO 主要组成部分如下：

（1）SDO 客户机。SDO 客户机使用 SDO 框架处理数据。SDO 客户机使用的不是特定于技术的 API 和框架，而是 SDO 编程模型和 API。SDO 客户机处理 SDO 数据图，不需要了解所处理的数据是如何持久保存或者序列化的。

（2）Data 中介服务。数据中介服务（DMS）负责从数据源创建数据图，依据数据图的变化更新数据源。数据中介框架不属于 SDO 1.0 规范。常见的 DMS 有 JDBC DMS、实体 EJB DMS 和 XML DMS 等。图 5-34 所示是 DMS 结构。

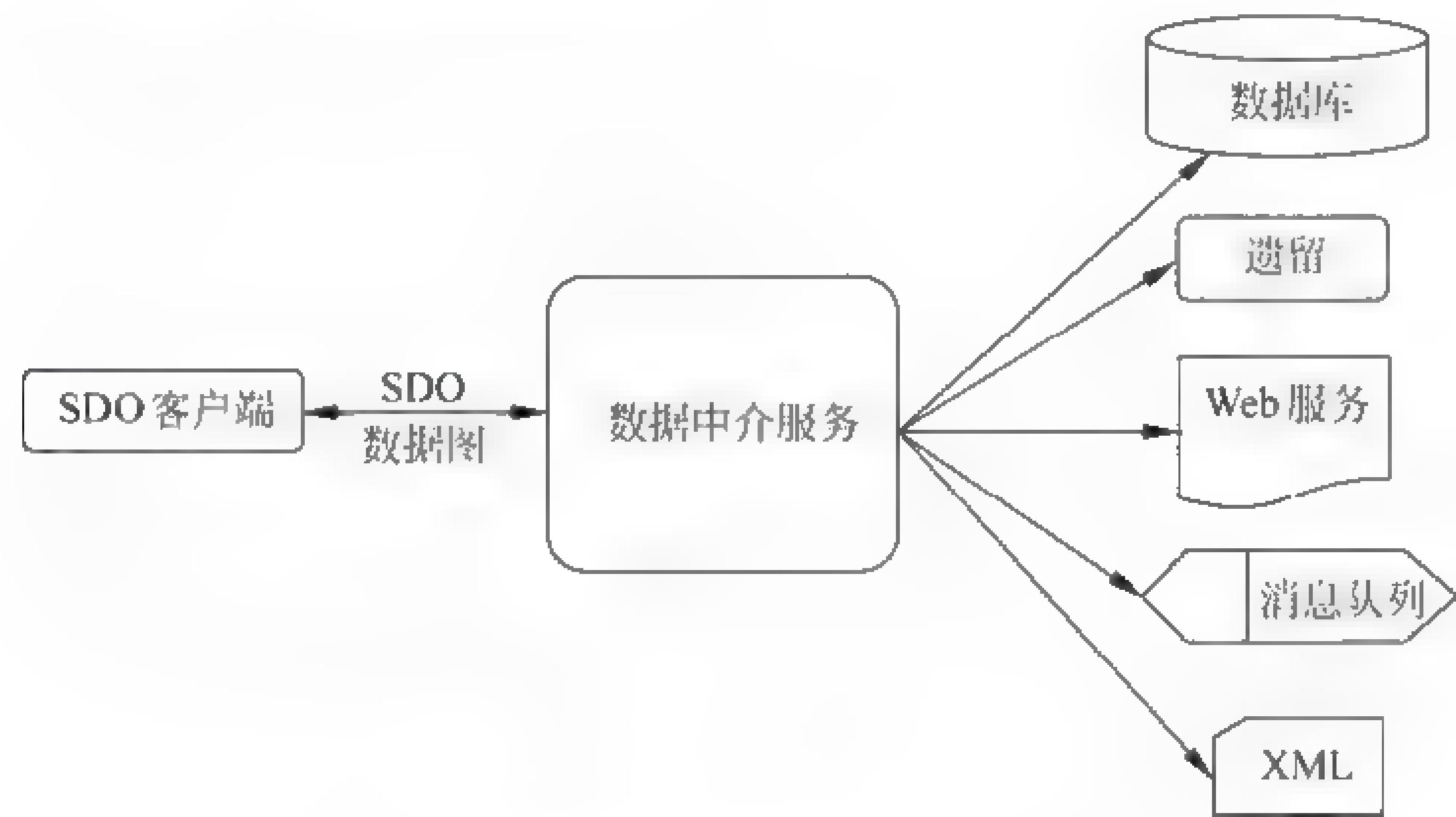

图 5-34　DMS 结构

（3）数据源。数据源不限于后端数据源（如持久存储数据库），是以自己的格式保存数据。只有 DMS 访问数据源，SDO 应用程序不访问数据源。SDO 应用程序可能只使用数据图中的数据对象。如下程序片段是数据图接口定义：

```
public interface DataGraph extends Serializable
{
    DataObject getRootObject();
    DataObject createRootObject(String namespaceURI, String typeName);
    DataObject createRootObject(Type type);
    ChangeSummary getChangeSummary();
    Type getType(String uri, String typeName);
}
```

下面继续介绍的每个组件对应于 SDO 编程模型的一个 Java 接口。SDO 参考实现提供了这些接口是基于 EMF（Eclipse Modeling Framework）的实现。

（1）数据对象。数据对象是 SDO 的基本组件。即是规范名称中的服务数据对象。数据对象是结构化数据的 SDO 表示。数据对象是通用的，它们提供了 DMS 创建的结构化数据的公共视图。比方说，虽然 JDBC DMS 需要知道持久性技术（比如关系数据库），以及如何配置和访问数据，而 SDO 客户机不需要了解这些细节。并且数据对象提供了易于使用的创建和删除方法（带有不同签名的 createDataObject()和 delete()）来获得自身类型（实例类、名称、属性和命名空间）的反射方法。使数据对象都链接在一起，并包含在数据图中。

（2）数据图。数据图提供了数据对象树的容器。数据图由 DMS 生成，供 SDO 客户机使用。当修改后，数据图被回传给 DMS 更新数据源。SDO 客户机可以遍历数据图，读取和修改数据图中的数据对象。SDO 是一种无连接的体系机构，因为 SDO 客户机与 DMS 和数据源没有连接，所以 SDO 客户机看到的只有数据图。此外，数据图可以包含表示不同数据源中数据的对象。数据图包含一个根数据对象，以及与根关联的所有数据对象和变更摘要（change summary）。当在应用程序组件（比如服务调用期间的 Web 服务请求者和提供者）之间进行传输、组件到 DMS 的传输（或者保存到磁盘）的时候，数据图被序列化为 XML。SDO 规范提供了序列化的 XML Schema。

（3）变更摘要。变更摘要包含在数据图中，表示对 DMS 返回的数据图的修改。变更摘要最初是空的（数据图刚返回客户机的时候），随着数据图的变化逐渐填充。在后台更新时，DMS 使用变更摘要将修改应用于数据源。变更摘要提供了数据图中被修改的属性（包括原来的值）、新增和删除的数据对象的列表，从而使 DMS 以递增方式高效地更新数据源。只有当变更摘要日志功能被激活时，才会将信息添加到数据图的变更摘要中。变更摘要提供了让 DMS 打开和关闭日志功能的方法，后面的例子中还将详细对其进行介绍。

（4）属性、类型和序列。数据对象用一系列属性保存其内容，每个属性都有一个类型，该类型既可以是如 in 的 t 基本类型，也可以是通用数据类型（如 Date），如果引用的话，还可以是其他数据对象类型。每个数据对象都为属性提供了访问和设置方法（getter 和 setter）。这些访问器方法有不同的重载版本，可以通过传递属性名（String）、编号（int）或者属性元对象本身来访问属性，其中 String 访问器还允许使用类 XPath 的语法访问属性，图 5-35 是 SDO 序列化结构图。

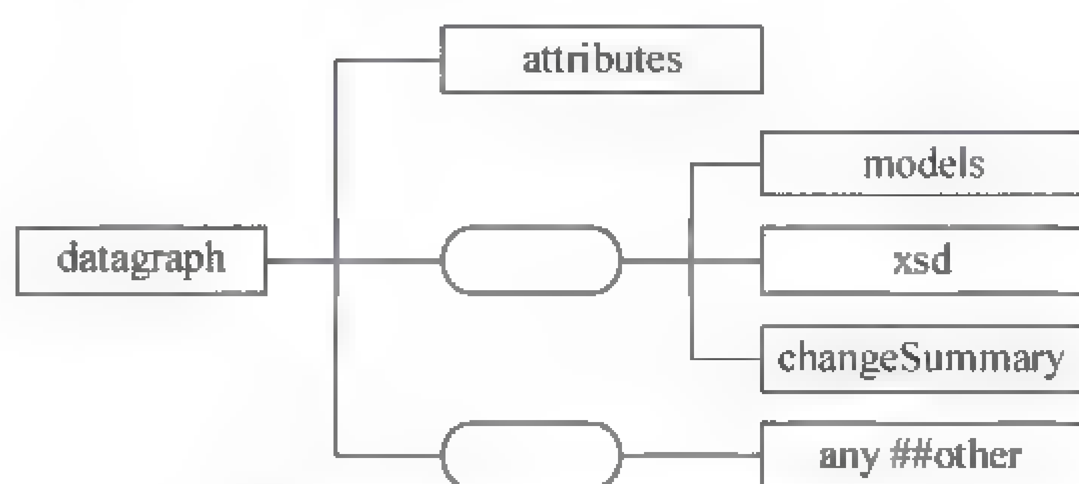

图 5-35　SDO 序列化结构图

服务数据对象通常用于将数据从一个应用程序传输到另一个应用程序。此用法模型可以方便地集成在 Web 服务环境中；在此环境中，客户机通常调用其他方提供的服务，并需要将数据从 Web 服务提供者传输到使用者。SDO 2.1 规范描述了使用静态或动态数据 API 创建

SDO 的两种方法。在两种情况下，表示 SDO 的对象都必须实现 commonj.sdo.DataObject 接口，通过该接口可与规范定义的其他 SDO API 进行交互。

数据源连接应用程序的任务是由数据中介服务完成的。客户机应用程序将查询数据中介服务，并从响应中得到数据图。客户机应用程序随后将更新后的数据图发送给数据中介服务，以将更新应用到原始数据源。此体系结构允许应用程序主要处理数据图和数据对象。

SDO 同时支持静态（或强类型）编程模型和动态（或松散类型）编程模型。这就提供了一个简单的编程模型，而不会牺牲各种工具和框架所需的动态模型。图 5-36 所示是 SDO 静态与动态访问模型。

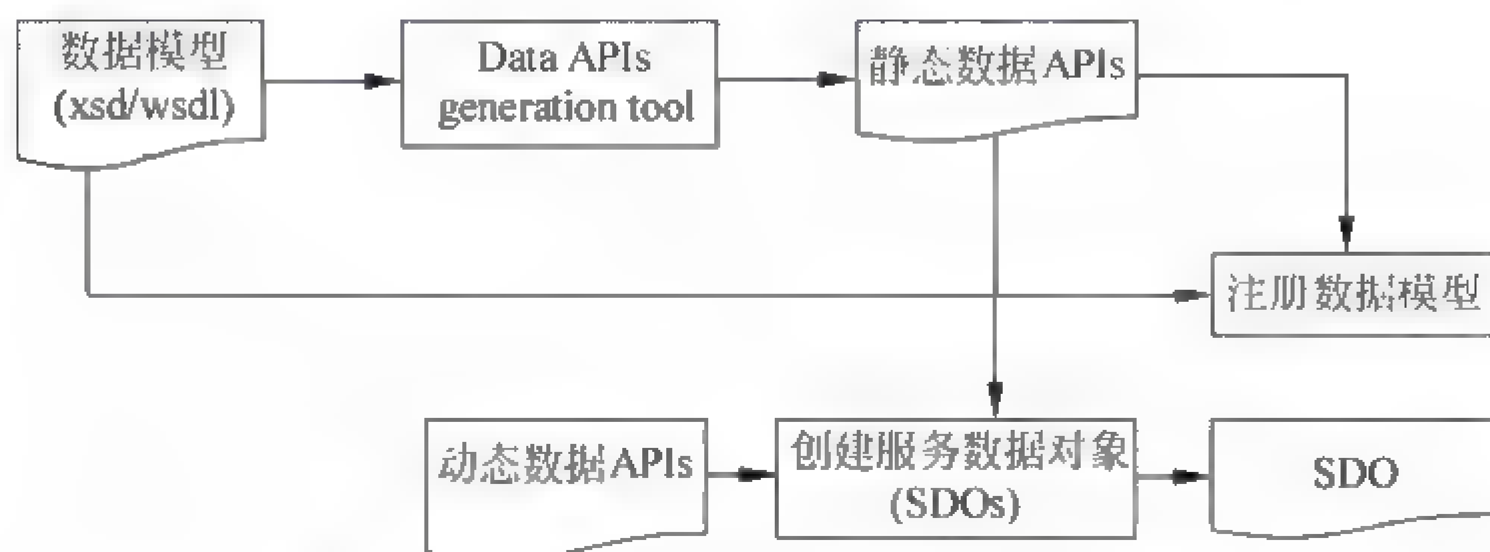

图 5-36　SDO 静态和动态数据 API 的用法

SDO 还提供了一个元数据 API，允许应用程序、工具和框架对数据图形的数据模型进行自检。SDO 元数据 API 对数据源特定的元数据 API 进行了统一，允许应用程序以统一的方式处理来自异类数据源的数据。并且 SDO 与语言无关，可在一系列编程语言中使用。简单地说，SDO 是一种数据应用程序开发框架，它包括一个体系结构和相应的 API。

2）SDO 应用描述

Service Data Object (SDO) 2.0 是一个开放标准数据模型编程 API，允许开发人员在较高的级别方便地操作数据；它简化和补充了 J2EE 的开发模式，提供了一种独特的访问异构数据源的 API。从而允许处理来自多种数据源的数据，其中包括关系数据库、实体 EJB 组件、XML 页面、Web 服务、Java Connector Architecture、JavaServer Pages 页面等。

当前的实现 SDO 2.0 的方法主要通过在 Eclipse 框架下使用 EMF（Eclipse Modeling Framework）2.2 SDK 完成，但这个 SDO 2.0 实现细节并不会影响开发人员根据新 API 编写程序。将来，开放源代码社区（通过 Apache Software Foundation）可能会决定提供不同的 SDO 2.0 实现，但这不应影响基于 SDO 2.0 API 构建的应用程序。同时，SDO2.0 API 优势的最基本方法是符合 XML 模式（XSD）的 XML 文档并对其进行读取操作。同时，要在不使用 SDO 2.0 的情况下完成相同的工作，开发人员需要理解 XML 解析器如何工作，并将数据解析逻辑与应用程序紧密集成。如果以后 XSD 需要更改，将需要对应用程序的各处进行调整，这可能对代码的质量带来灾难性的影响，图 5-37 所示是基于 SDO 的应用体系结构，这种体系结构使 SDO 具有以下几个优势：

（1）简化数据访问。为多种企业信息系统（EIS）提供统一的数据访问，包括数据库、遗留应用程序（使用 JCA）、XML 或者是 Web 服务数据源；并通过使用 SDO 的一种独特而简单的模型，应用程序摆脱了使用多种 API 和框架进行数据访问的复杂工作。

（2）数据提取。使用 SDO 后，数据的表示是独立于其数据源的，它采用了一种叫做 Domain

Store 的 J2EE 模式，这种级别的数据提取有很多优点，例如使数据操作变得更容易，实现了不同层之间的松耦合。

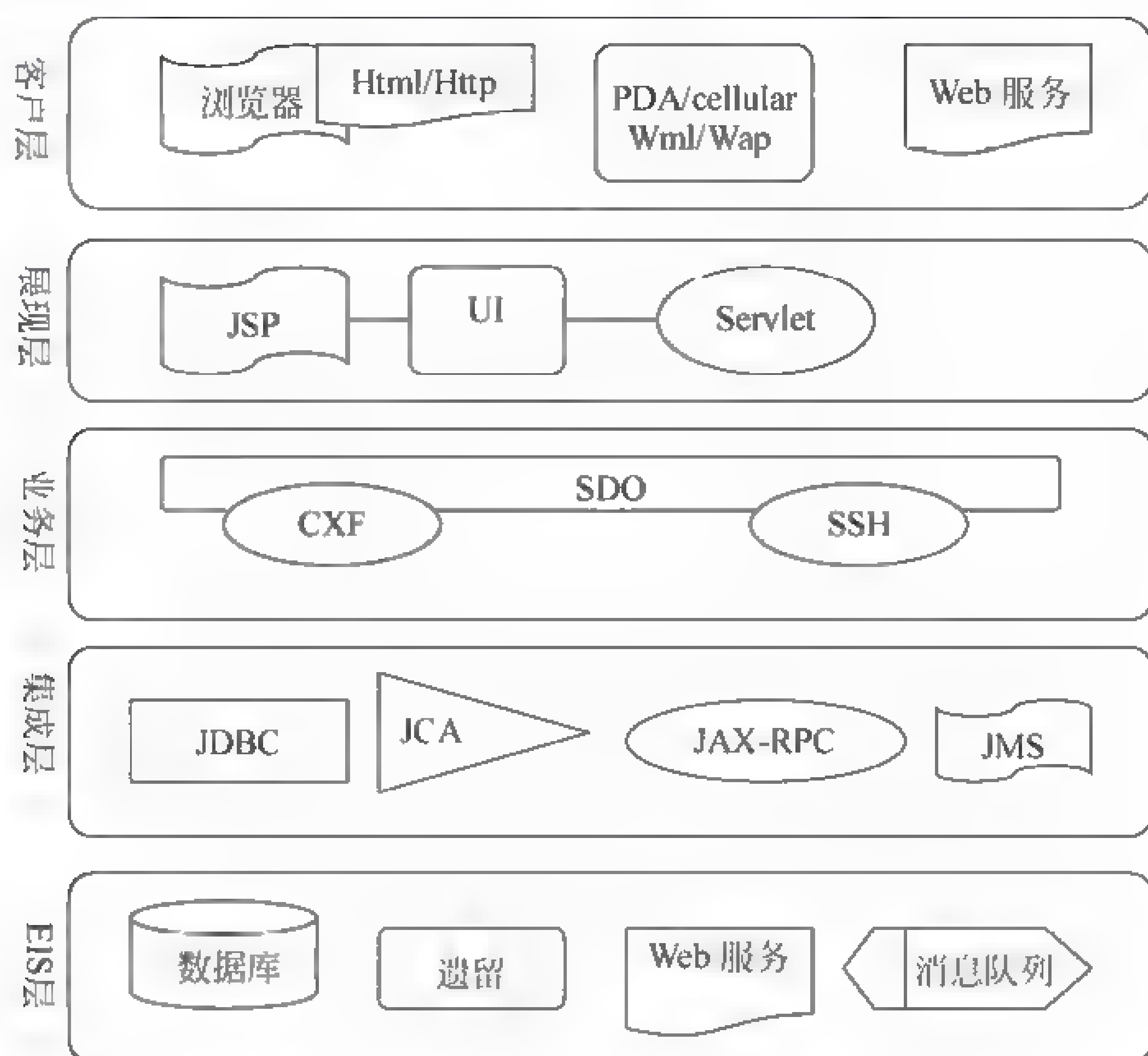

图 5-37　基于 SDO 的应用体系结构

（3）数据操作。一旦检索到信息后，SDO 会提供一种统一的编程语言进行数据操作，简单的说，就是通过使用 API 及其接口，SDO 客户机可以读取数据和修改数据。SDO 为此提供了连接和断开连接的两种模型。

（4）数据传输。SDO 有一部分概念是关于传输对象（Transfer Object）和传输对象组装程序（Transfer Object Assembler）模式的。数据封装到 SDO 对象中后，它就可以在 J2EE 层间高效地传输。

（5）设计模式的采用。SDO 的一个关键目标是鼓励大家采用公用的 J2EE 模式，这也是 SDO 体系结构以一些广为人知的模式为基础的原因，例如传输对象（Transfer Object）、数据访问对象（Data Access Object）、传输对象组装程序和 Domain Store 等。如果使用了 SDO，应用程序就可以从这些经过了验证的设计策略中受益，从而可以推动分层技术和松耦合的发展。

实现 SDO 的 EMF 当前版本是 2.5。通常 EMF 安装成功后，运行时只需要两个插件：org.eclipse.emf.commonj.sdo 和 org.eclipse.emf.ecore.sdo，就可以完成 SDO 的应用。而 EMF 代表 Eclipse Modeling Framework（Eclipse 建模框架），它根据使用 Java 接口、XML Schema 或者 UML 类图定义的数据模型生成统一的元模型（称为 Ecore），这时可以将元模型与框架结合在一起使用，从而创建高质量的模型实现。同时，EMF 提供了持久性、一个有效的反射类属对象操纵 API 和一个变更通知框架。EMF 还包括用来构建 EMF 模型编辑器的一般的重用类。如下程序片段是 SDO 为 Web 服务完成 WSDL 的例子。其详细介绍请参见 SDO 规范文档说明。程序清单 5-34 是 SDO 为 Web 服务完成 WSDL 的例子。

程序清单 5-34：

```
<wsdl:definitions name="Name"
targetNamespace="http://example.com"
xmlns:tns="http://example.com"
xmlns:company="company.xsd"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
Page 144
<wsdl:types>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="company.xsd"
xmlns:company="company.xsd"
xmlns:sdo="commonj.sdo"
elementFormDefault="qualified">
<element name="companyDatagraph" type="company:CompanyDataGraphType"/>
<complexType name="CompanyDataGraphType">
<complexContent>
<extension base="sdo:BaseDataGraphType">
<sequence>
<element name="company" type="company:CompanyType"/>
</sequence>
</extension>
</complexContent>
</complexType>
</schema>
</wsdl:types>
<wsdl:message name="fooMessage">
<wsdl:part name="body" element="company:companyDataGraph"/>
</wsdl:message>
<wsdl:message name="fooResponseMessage"></wsdl:message>
<wsdl:portType name="fooPortType">
<wsdl:operation name="myOperation">
<wsdl:input message="tns:fooMessage"/>
<wsdl:output message="tns:fooResponseMessage"/>
</wsdl:operation>
</wsdl:portType>
<wsdl:binding name="fooBinding" type="tns:fooPortType">
<soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
<wsdl:operation name="myOperation">
<soap:operation/>
<wsdl:input>
```

```
<soap:body use "literal"/>
</wsdl:input>
<wsdl:output>
<soap:body use="literal"/>
</wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="myService">
<wsdl:port name="myPort" binding="tns:fooBinding">
<soap:address location="http://localhost/myservice"/>
</wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

5.6.5　BPEL4WS

前面章节已经描述了 BPEL 及与 SOA 的内容和基本应用方法，本节继续叙述 BPEL4WS 与 BPEL 的关系，它是专为整合 Web Services 而制定的一项规范标准，BPEL 的作用是将一组现有的服务组合起来，从而定义一个新的 Web 服务。因此，BPEL 基本上是一种实现此种组合的语言，而组合服务的接口也被描述为 WSDL portType 的集合。BPEL4WS（即 BPEL，读作 bee-pel）是 IBM、Microsoft 和 BEA 等三家主要的企业应用程序服务器供应商共同努力的成果。BPEL4WS 定义了描述业务流程的 XML 语法，它的关键之处在于 Web 服务之间的 peer-to-peer 交互的表示法，Web 服务是用 WSDL 进行描述的，而 WSDL 是 Web 服务描述的事实标准。流程本身及其合作伙伴都模型化为 WSDL 服务，而使用 BPEL4WS 语法来创建流程定义。根据 WFMC 定义，流程定义是以支持自动化操作（如通过工作流管理系统进行建模或加以实现）的形式表示业务流程的方法。流程定义确定了如何协调流程实例及其合作伙伴之间的交互。

BPEL4WS 作为可执行流程的实现语言，它的作用是将一组现有的服务整合起来，从而定义一个新的 Web 服务。因此，BPEL4WS 基本上是一种实现这样的整合的语言。与其他任何 Web 服务一样，整合服务的接口也被描述为 WSDL portType 的集合。整合（称为流程）指明了服务接口与整合的总体执行的配合情况，图 5-38 是 BPEL4WS 流程实现的 Web 服务视图。

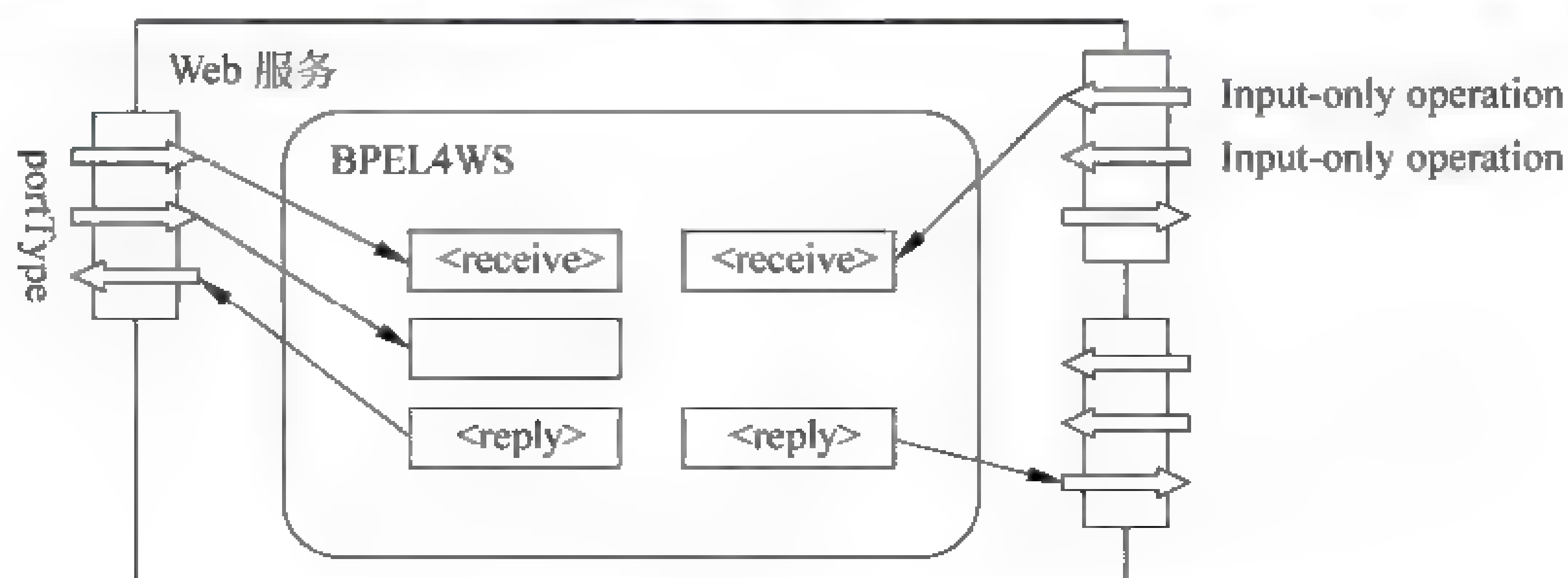

图 5-38　BPEL4WS 流程实现的 Web 服务视图

5.7 语义描述语言

随着需求对获取数据精确性的提高，以及数据量的增加，单纯基于XML的数据描述和表示方法已不能满足用户的需求和企业的要求；也不能实现数据的自动识别和快速搜索。在这种背景下，语义化的方法逐渐受到人们关注，且逐渐应用到了相关领域。下面就从RDF和语义Web服务两角度进行描述语义语言。

5.7.1 RDF

资源描述框架（Resource Description Framework，RDF），是万维网联盟（W3C）提出的一组置标语言的技术标准，以便更为丰富地描述和表达网络资源的内容与结构。即它是一个用于表达关于万维网上的资源的信息的语言。它专门用于表达关于Web资源的元数据，比如Web页面的标题、作者和修改时间，Web文档的版权和许可信息，某个被共享资源的可用计划表等。然而，将“Web资源（Web resource）”这一概念一般化后，RDF可被用于表达关于任何可在Web上被标识的事物的信息，即使有时它们不能被直接从Web上获取。比如关于一个在线购物机构的某项产品的信息（例如关于规格、价格和可用性信息），或者是关于一个Web用户在信息递送方面的偏好的描述①。即RDF也是一种描述有关Web资源的格式化语句集合的模型，还可将RDF看作Web的元数据系统。RDF语句在概念上分为三部分，每条语句包含一个主题（一个URI）；一个谓词（也是一个URI）；还包括一个对象（一个URI或字母数据值），如下程序片段就是RDF的表示方法。

```
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:ex = "http://example.com#"
   xmlns:dc="http://purl.org/dc/elements/1.1/#">
   <rdf:Description rdf:about="http://www.chaosmagnet.com">
      <ex:name>Chaos Magnet</ex:name>
      <dc:creator rdf:resource="http://www.nicholaschase.com" />
   </rdf:Description>
</rdf:RDF>
```

XML不是表示RDF的唯一方式，但XML是目前最流行的方式之一。RDF最常见用途是描述已经存在的资源，比如整个或部分RSS提要。在这些情况下，RDF通常包含在另一个XML文档中，并且通过使用XML命名空间来分辨。同时，RDF描述如何指定关于资源的信息，但是信息本身没有任何含意。要定义项的类型及其他们之间的关系（比如类和子类），需要向词汇表中添加一些具体的表达对象，添加的这些对象都封装在RDF Schema中。RDF Schema（又称RDFs）具有它自己的命名空间和很多预定义的元素与属性。如下程序片段是RDFs基本类：

① http://zh.transwiki.org/cn/rdfprimer.htm

```
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xml:base="http://www.example.com/mashup/">
  <rdf:Description rdf:ID="Service">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  </rdf:Description>
  <rdf:Description rdf:ID="SOAPService">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  </rdf:Description>
  <rdf:Description rdf:ID="RESTService">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  </rdf:Description>
  <rdf:Description rdf:ID="RSSFeed">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  </rdf:Description>
</rdf:RDF>
```

同样，还可以在这个程序片段下创建与之相关的子类、个体、属性和子属性等。

1. 基于 XQuery 的 XML 与 RDF 的转换

XQuery 起源于 1998 年，是由 W3C 发起与制定的查询语言。对于 XML 用户来说，最熟悉的 XQuery 关键组件是 XPath，它本身也是一个 W3C 规范。虽然 XPath 和 XQuery 都能实现一些相同的功能，但是 XPath 比较简洁而 XQuery 更加强大和灵活。对于很多查询来说 XPath 非常合适。在数据方面，XQuery 具有类似于 SQL 的外观和能力。它们之间的对比如表 5-8 所示。

表 5-8　Xquery 与 SQL 的比较

XQuery	SQL
支持路径表达式，使程序员可以在 XML 的层次结构中导航	不支持路径表达式
支持有类型的和无类型的数据	总是用特定类型进行定义
不存在 null 值，因为 XML 文档忽略没有的或未知的数据	使用 null 值表示没有的或未知的数据值
返回 XML 数据序列	返回不同 SQL 数据类型的结果集

XSLT（Extensible Stylesheet Language Transformation）和 XQuery 的相同之处主要表现在：

（1）都输入 XML，输出 XML；

（2）具有相同的数据模型和类型系统；

（3）都是声明性的，即没有副作用；

（4）都使用 Xpath 时，具有相同的内置函数库。

当然也有明显的区别，主要表现在：

（1）XQuery 非常适合查询 XML；

（2）XQuery 适合控制处理（比如 XSLT），特别是在 XML 数据库中；

（3）在 XQuery 中遍历非常麻烦；

（4）使用 XSLT 模板匹配更合适；

（5）需要格式化数字时，XSLT 更方便。也可以说 XQuery 代表 XSLT 2.0 的一个功能子集。比如，XQuery 没有动态邦定的概念，而 XSLT 通过模板匹配规则提供了动态匹配。XSLT 还提供了某种形式的多态，即使用 xsl:import 覆盖模板匹配规则，而 XQuery 没有这样的设施。

因此，XQuery 采用一种简单的语法，混合了 XML、XPath、注释、函数以及将其结合在一起的专用表达式语法。XQuery 代码完全由表达式组成，没有语句，所有的值都是序列。Xquery 也是一种高端的、强类型的函数语言，非常适合表达从 XML 文档或者大型 XML 存储库（repository）中获取数据的查询。而 XPath 是一种领域专用语言（Domain-Specific Language，DSL），并很快成为了其他更通用语言的重要组成部分。编程语言通过模块和类把 XPath 结合进来，有时候甚至直接进入语言的语法。这种情况和以前正则表达式的情况类似。如下程序片段是 Xquery 语法示例代码（表示扫描查询文档中所有图书的查询）：

```
<result>
 {
  for $i in doc("rss.xml")/rss/channel/item
  where starts-with($i/pubDate/text(),"Fri")
  return
    <friday>
      { $i/title/text() }
    </friday>
 }
 {
  for $i in doc("rss.xml")/rss/channel/item
  where contains($i/title/text(),"XMI")
  return
    <xmi>
      { $i/title/text() }
    </xmi>
 }
</result>
```

XML 和 RDF 是两种不同的结构化数据方法，一种方法使用的组织原则和概念不一定能完全转化成另一种方法所使用的组织原则和概念（下面以一个 bib.xml 的书目为例进行描述）。要实现 XML 与 RDF 转换，需要理解 RDF/XML 中的 Striping 概念，Striping 是嵌套 RDF/XML 中交替出现的元素类型来模拟 RDF 图中的路径浏览（path traversal）的一种方法。对于一般的 RDF 路径，资源与谓词交替出现的方式为：resource -> predicate -> resource -> predicate -> resource。路径的这种 RDF/XML 序列化嵌套模式精确反映了元素的交替。在一般的形式中，表示 RDF 资源的<rdf:Description>元素和表示这些资源谓词的特定词汇表元素是交替出现的，对于该实例的 RDF 路径如下：

```
bibResource ->
 bookPredicate ->
  bookResource ->
```

```
    authorPredicate ->
     authorResource ->
      lastnamePredicate ->
```

则表示这条路径的 RDF/XML 如下：

```
<rdf:Description rdf:about="...">     // a named bib resource
 <bibterms:book>                      // a book predicate
  <rdf:Description>                   // an anonymous book resource
   <bibterms:author>                  // an author predicate
    <rdf:Description>                 // an anonymous author resource
     <bibterms:last>                  // a last name predicate
      ...
```

完成了 XML 与 RDF 转换，需要 XML 与 RDF 转换的解析工具：XQuery 处理程序和 W3C RDF Validation Service。其中 XQuery 处理程序一般包括基于开放源代码的、Java 语言的 XQuery 处理程序的 Saxon，使用不需要安装的、基于 Web 的联机系统的 Galax。RDF Validation Service 接受 RDF/XML（RDF 的序列化形式）解析，并确定它是否为有效的 RDF。它还生成 RDF 原生三元组格式的序列化，还能够生成（可选）数据的 RDF 图。当在实现 XML 与 RDF 转换时，首先需要创建一个空的外部<rdf:RDF>元素包装器，然后填充所要转换内容。例如：

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
</rdf:RDF>
```

（1）Saxon。若希望找到一个高度符合 XQuery 规范并且能够处理所选数据文件的工具，那么 Saxon[①]是一个很好的选择，它可以从 SourceForge 下载，当前版本是 9.3.05。能全面支持 XSLT，Xpath，XQuery 和 XML Schema。

（2）Galax。Galax 是运行 XQuery 查询的另一种选择。可以直接联机使用它，不必进行本地安装。如果想明白 XQuery 处理程序的发展情况，那么 Galax 还有另一个优于 Saxon 的地方。就是 Galax 的作者积极参与了为 XQuery 提供数学基础的形式化语义的研究。Galax 还提供了几种基于 HTML 的转换，允许查看查询计算过程中的中间结果。

（3）RDF Validator。W3C 的在线 RDF Validation Service 常常被称为 RDF Validator，主要提供了检查 RDF/XML 的合法性、查看从 RDF/XML 解析得到的 RDF 三元组，以及还可以选择生成 RDF 结果图。

最终转换结果的程序片段如下：

转换前的 XML 文档：

```
<bib>
   <book year="1994">
      <title>TCP/IP Illustrated</title>
      <author><last>Stevens</last><first>W.</first></author>
```

① http://saxon.sourceforge.net/

```
      <publisher>Addison Wesley</publisher>
      <price> 65.95</price>
   </book>
  <book year="1992">
      <title>Advanced Programming in the Unix environment</title>
      <author><last>Stevens</last><first>W.</first></author>
      <publisher>Addison-Wesley</publisher>
      <price>65.95</price>
   </book>
    <book year="2000">
      <title>Data on the Web</title>
      <author><last>Abiteboul</last><first>Serge</first></author>
      <author><last>Buneman</last><first>Peter</first></author>
      <author><last>Suciu</last><first>Dan</first></author>
      <publisher>Morgan Kaufmann Publishers</publisher>
      <price>39.95</price>
   </book>
    <book year="1999">
      <title>The Economics of Technology and Content for Digital TV</title>
      <editor>
         <last>Gerbarg</last><first>Darcy</first>
         <affiliation>CITI</affiliation>
      </editor>
      <publisher>Kluwer Academic Publishers</publisher>
      <price>129.95</price>
   </book>
</bib>
```

基于 Saxon 的 XML 与 RDF 转换结构：

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:bibterms="http://www.book-stuff.com/terms/"
     xmlns:dc="http://purl.org/dc/elements/1.0/"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
   <rdf:Description rdf:about="http://www.book-stuff.com/bib">
      <bibterms:book rdf:parseType="Resource">
         <bibterms:year>1994</bibterms:year>
         <dc:title>TCP/IP Illustrated</dc:title>
         <bibterms:author>
            <bibterms:last>Stevens</bibterms:last>
            <bibterms:first>W.</bibterms:first>
         </bibterms:author>
         <bibterms:publisher>Addison-Wesley</bibterms:publisher>
         <bibterms:price> 65.95</bibterms:price>
      </bibterms:book>
      <bibterms:book rdf:parseType "Resource">
```

```
        <bibterms:year>1992</bibterms:year>
        <dc:title>Advanced Programming in the Unix environment</dc:title>
        <bibterms:author>
          <bibterms:last>Stevens</bibterms:last>
          <bibterms:first>W.</bibterms:first>
        </bibterms:author>
        <bibterms:publisher>Addison-Wesley</bibterms:publisher>
        <bibterms:price>65.95</bibterms:price>
      </bibterms:book>
      <bibterms:book rdf:parseType="Resource">
        <bibterms:year>2000</bibterms:year>
        <dc:title>Data on the Web</dc:title>
        <bibterms:author>
          <bibterms:last>Abiteboul</bibterms:last>
          <bibterms:first>Serge</bibterms:first>
        </bibterms:author>
        <bibterms:author>
          <bibterms:last>Buneman</bibterms:last>
          <bibterms:first>Peter</bibterms:first>
        </bibterms:author>
        <bibterms:author>
          <bibterms:last>Suciu</bibterms:last>
          <bibterms:first>Dan</bibterms:first>
        </bibterms:author>
        <bibterms:publisher>Morgan Kaufmann Publishers</bibterms:publisher>
        <bibterms:price>39.95</bibterms:price>
      </bibterms:book>
      <bibterms:book rdf:parseType="Resource">
        <bibterms:year>1999</bibterms:year>
        <dc:title>The Economics of Technology and Content for Digital TV
        </dc:title>
        <bibterms:editor>
          <bibterms:last>Gerbarg</bibterms:last>
          <bibterms:first>Darcy</bibterms:first>
        </bibterms:editor>
        <bibterms:publisher>Kluwer Academic Publishers</bibterms:publisher>
        <bibterms:price>129.95</bibterms:price>
      </bibterms:book>
    </rdf:Description>
  </rdf:RDF>
```

2. 将 RDF 引入到 WSDL 中，增强 WSDL 的语义性

RDF 是 W3C 用于定义 XML 对象的元数据的正式格式。它与 WSDL 很相似，从概念上讲，它也是有关基于 XML 的服务的元数据集合。WSDL 规范提供了一个基于 XML 的简单语汇表，用来描述基于 XML 的 Web 服务。这些服务本身使用 SOAP、HTTP、SMTP（Simple

Message Transfer Protocol，简单邮件传输协议）或通过其他方式进行通信，而 WSDL 为用户提供设置这些通信所需的元数据。但是 WSDL 本身不规定如何发布或公布这种服务描述，而是将这项任务留给其他规范。UDDI 是用来创建 Web 服务目录的一个倡议，它定义了对 WSDL 描述进行编目和调度的一个框架，但是它刚刚出现而且相当复杂，如下程序片段就是 RDF 与 WSDL 转换结构：

```
<?xml version="1.0"?>
 <definitions name="EndorsementSearch"
   targetNamespace="http://namespaces.snowboard-info.com"
   xmlns:es="http://www.snowboard-info.com/EndorsementSearch.wsdl"
   xmlns:esxsd="http://schemas.snowboard-info.com/EndorsementSearch.xsd"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:w="http://schemas.xmlsoap.org/wsdl/"
   xmlns="http://schemas.xmlsoap.org/wsdl/">
   <types>
     <schema targetNamespace=" "
           xmlns="http://www.w3.org/1999/XMLSchema">
       <element name=" ">
         <complexType>
           <sequence>
             <element name=" " type="string"/>
             <element name=" " type="string"/>
             <element name=" " type="string"/>
           </sequence>
         </complexType>
       </element>
       ......
     </schema>
   </types>

   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
     <message w:name=" " rdf:ID=" ">
       <part w:name="body" w:element="esxsd: "/>
     </message>
     ......
     <portType w:name=" "  rdf:ID=" ">
       <operation w:name=" ">
         <input rdf:resource=" "/>
         <output rdf:resource=" "/>
         <fault rdf:resource=" "/>
       </operation>
     </portType>

     <binding w:name " "  w:type " " rdf:ID " ">
       <soap:binding soap:style "document"
```

```
                    soap:transport "http://schemas.xmlsoap.org/soap/http"/>
      <operation rdf:about " ">
        <soap:operation
          soap:soapAction=" "/>
        <input>
          <soap:body soap:use="literal" soap:namespace="*.xsd"/>
        </input>
        <output>
          <soap:body soap:use="literal" soap:namespace="*.xsd"/>
        </output>
        <fault>
          <soap:body soap:use="literal" soap:namespace="*.xsd"/>
        </fault>
      </operation>
    </binding>

    <service w:name=" " rdf:ID=" ">
      <documentation> 格式说明 </documentation>
      <port rdf:about=" ">
        <binding rdf:resource=" "/>
        <soap:address soap:location=" "/>
      </port>
    </service>
  </rdf:RDF>
</definitions>
```

5.7.2　OWL-S

Web Ontology Language（OWL）是一种 RDF 应用程序，通常使用 RDF/XML 编码，它添加了一种丰富的词汇表，可以用来按照格式分类并分析 RDF 资源。SPARQL Query Language for RDF 是用于查询 RDF 数据的特殊语法，Gleaning Resource Descriptions from Dialects of Languages（GRDDL）是从 XML 文档中提取 RDF 数据的系统。随着微格式在 Web 中的日益流行，GRDDL 的地位开始变得重要起来。另一种与 RDF 相关的规范适合用于微格式，即 RDFa Syntax，该规范是一组特殊的属性集合，可用于将 RDF 数据嵌入到诸如 XHTML 这样的 XML 格式中。而 OWL-S（Ontology Web Language for Services）[①]是在 OWL 基础上提供一种支持 Web 服务描述的语义化的本体语言。OWL-S 包括三个组件：ServiceProfile、ServiceModel 和 ServiceGrounding，如图 5-39 所示，可以说 WSDL 与 UDDI 使 Web 服务实现了自动化，OWL-S 使得 Web 服务实现智能化。

但 WSDL 与 OWL-S 具有三个共同的目标：

（1）一个 OWL-S 原子流程与一个 WSDL 操作相符合；

（2）OWL-S 原子流程的输入、输出与 WSDL 信息相符合；

---

① http://www.w3.org/Submission/OWL-S/

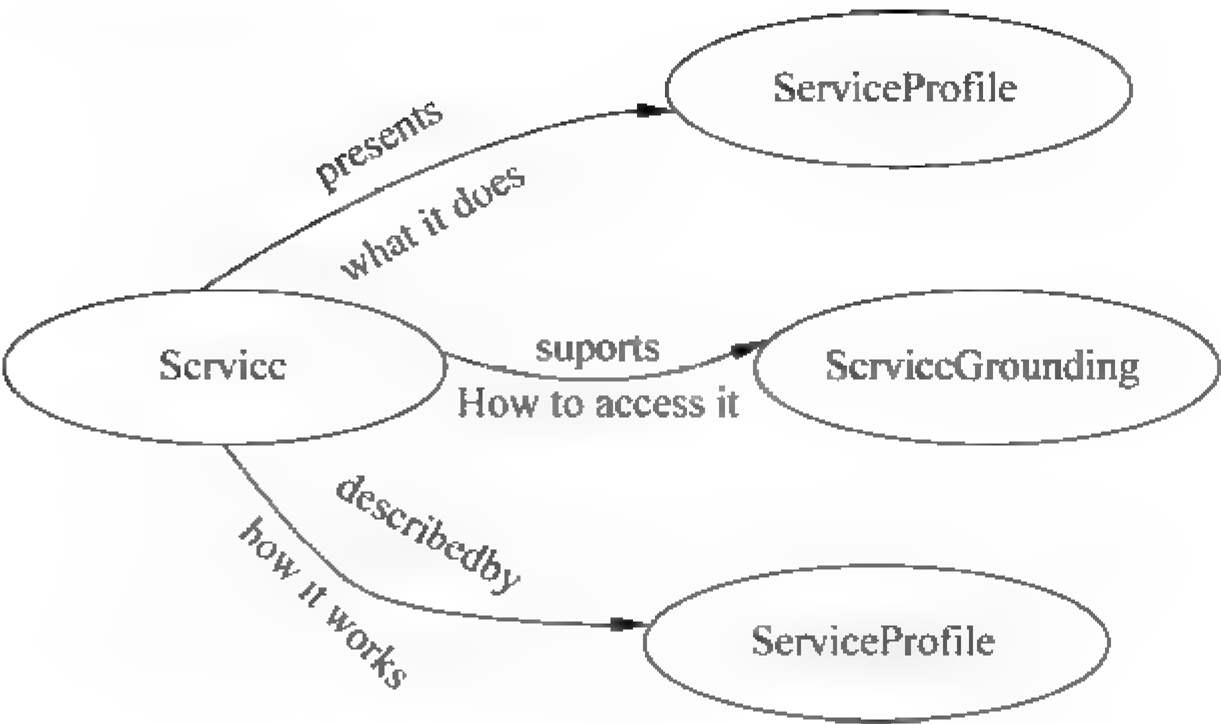

图 5-39 OWL-S 组成结构

（3）OWL-S 原子流程的输入、输出类型与 WSDL 摘要类型相符合。

图 5-40 是 WSDL 与 OWL-S 对应映射结构图。

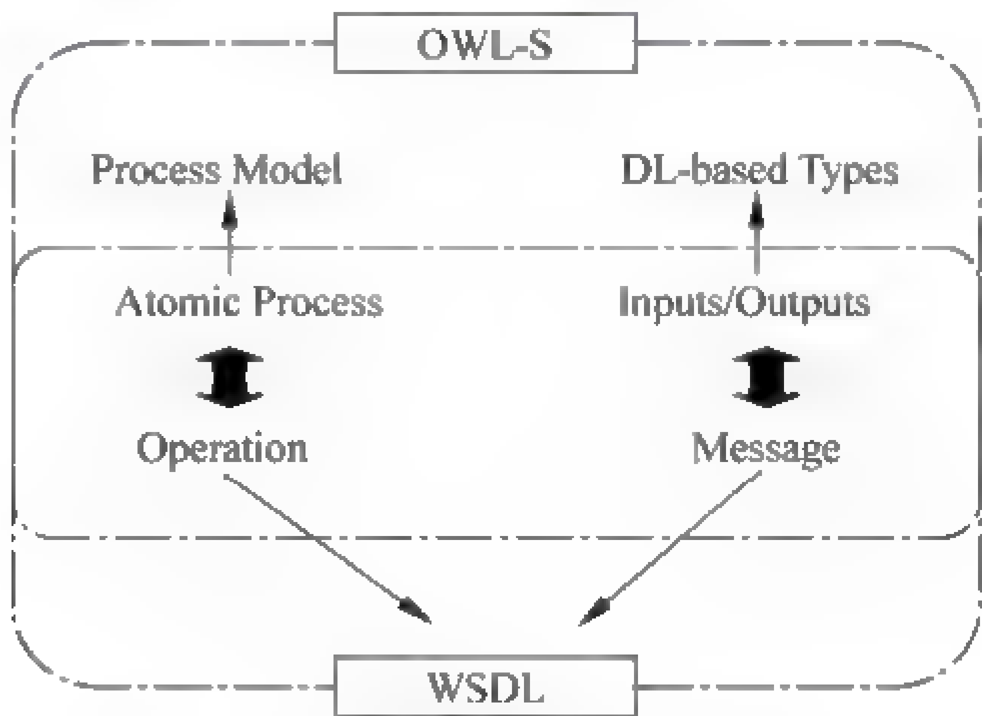

图 5-40 WSDL 与 OWL-S 映射结构

下面以一个在线书店为例为进一步说明 OWL-S 的应用方法，如图 5-41 所示。

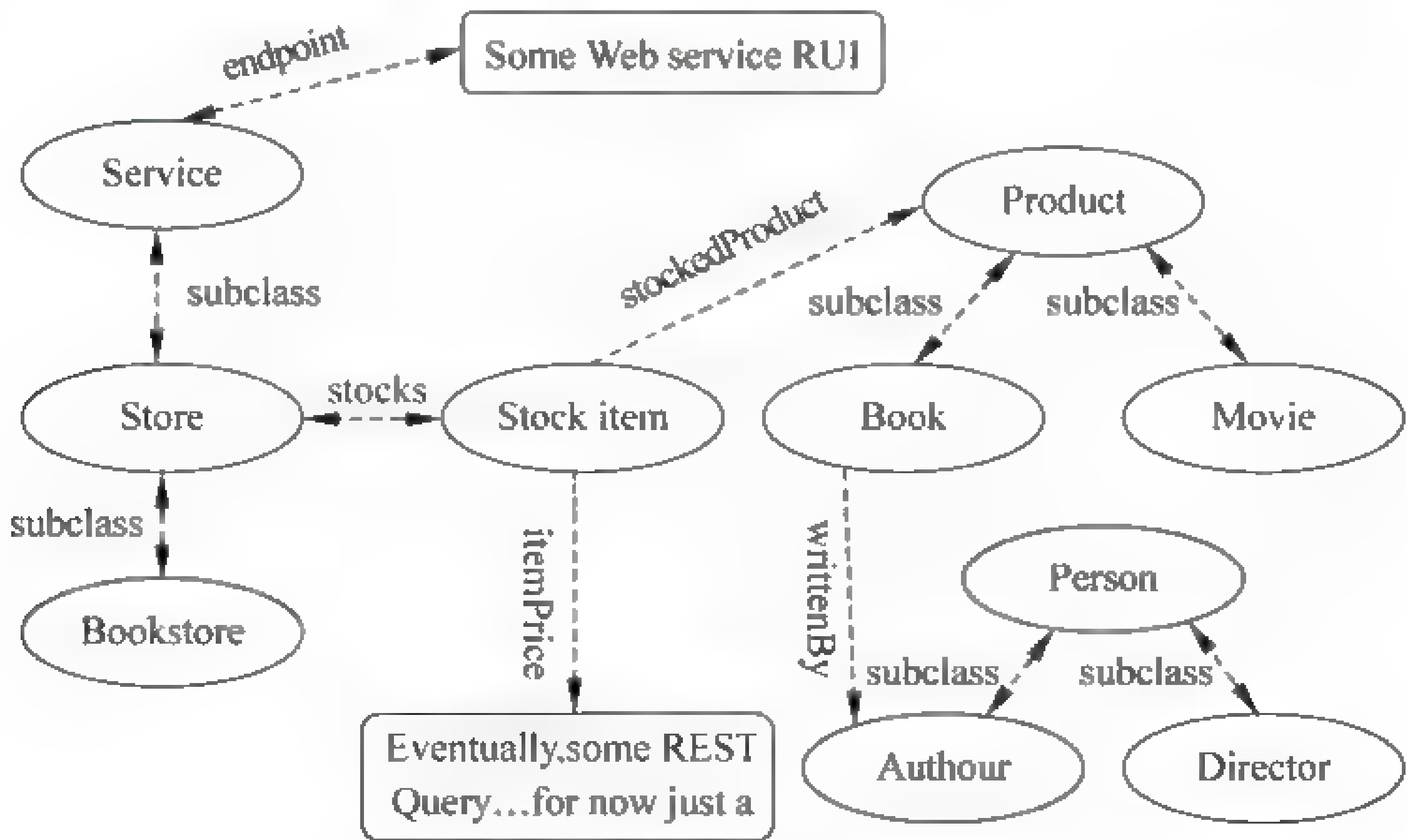

图 5-41 在线书店的本体结构

从图中的左上角开始，首先是 Service，然后是它的子类 Store 及 Store 子类 Bookstore。

商店里存有 StockItems，每个项都有 itemPrice 并对应某个 stockedProduct。对该例来说，产品可以是 Book 或 Movie。如果是 Book，则必须至少有一个 Author。Author 是 Person 的子类，Director 也是它的子类。如下程序片段就是创建一个简单本体的方法：

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
 <!ENTITY store "http://example.com/stores#" >
 <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
 <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >]>
<rdf:RDF
 xmlns = "http://example.com/store#"
 xmlns:store = "http://example.com/store#"
 xml:base = "http://example.com/store#"
 xmlns:owl = "http://www.w3.org/2002/07/owl#"
 xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdfs= "http://www.w3.org/2000/01/rdf-schema#"
>
<owl:Ontology rdf:about="">
 <rdfs:comment>
 一个在基于 OWL-S 的在线书店的例子
 </rdfs:comment>
 <rdfs:label>BookStore Ontology</rdfs:label>
</owl:Ontology>
 </rdf:RDF>
```

在上面程序片段中的第二步为每个主要的 XML 命名空间字符串定义了 XML 实体。这些实体可以在需要的时候方便地添加命名空间。比方说，如果指定了 XML Schema 定义的数据类型，可以使用 &xsd。同时在第二步定义了 RDF 元素本身和所有必要的命名空间。其中包括 owl:Ontology 元素。第三步表示 owl: namespace（http://www.w3.org/2002/07/owl#）中的某个地方定义了类 Ontology，该定义允许创建单个的 Ontology 对象，因为它是主元素，不需要引用其他任何资源。

下面对 OWL-S 的基本属性进行进一步分析与总结：

（1）在 OWL 中，rdf:Class 表示具体的类，然后将其作为类型使用，并通过该类可以创建这些个体类型的子类。owl:Thing 看作是其他一切的基类。owl:Nothing 代表没有任何成员的空类。如下程序片段就是创建类的基本方法：

```
<owl:Class rdf:ID="Store">/*创建类*/
 <rdfs:subClassOf rdf:resource="#Service"/>/*创建子类*/
 <rdfs:label>Online Store</rdfs:label>
</owl:Class>
```

（2）在 OWL 中，它与 RDF 和 RDFs 是不同的，即 OWL 有两种类型的属性：DatatypeProperty 和 ObjectProperty。其中 DatatypeProperty 定义的属性的值必须是简单类型，ObjectProperty 则以对象（或者某个类的个体）作为值。程序片段如下：

```
<owl:DatatypeProperty rdf:ID="title">
 <rdfs:domain rdf:resource="#Book"/>
 <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
......
<owl:ObjectProperty rdf:ID="genreOf">
 <rdfs:domain rdf:resource="#Book"/>
 <rdfs:range rdf:resource="#BookGenre"/>
</owl:ObjectProperty>
```

（3）验证本体一般可使用任何 OWL 推理器（reasoner）。推理基本性质主要包括传递性、对称性、逆向性、等价、等价属性和同一性。下面是各推理性质的程序片段：

传递性：

```
<owl:Class rdf:ID="ProductCategory" />
<owl:TransitiveProperty rdf:ID="isSubcategoryOf">
 <rdfs:domain rdf:resource="#ProductCategory"/>
 <rdfs:range rdf:resource="#ProductCategory"/>
</owl:TransitiveProperty>
<ProductCategory rdf:ID="Books"/>
<ProductCategory rdf:ID="Arts and Entertainment">
 <isSubcategoryOf rdf:resource="#Books" />
</ProductCategory>
<ProductCategory rdf:ID="Comics and Graphic Novels">
 <isSubcategoryOf rdf:resource="#Arts and Entertainment" />
</ProductCategory>
```

对称性：

```
<owl:SymmetricProperty rdf:ID="bundledWith">
 <rdfs:domain rdf:resource="#Product"/>
 <rdfs:range rdf:resource="#Product"/>
</owl:SymmetricProperty>
```

逆向性：

```
<owl:ObjectProperty rdf:ID="writtenBy">
 <rdfs:domain rdf:resource="#Book"/>
 <rdfs:range rdf:resource="#Author"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="writerOf">
 <owl:inverseOf rdf:resource="#writtenBy"/>
</owl:ObjectProperty>
```

逆向性有一个 InverseFunctionalProperty 属性，它能有效地简化数据的聚合，例如下面的程序片段：

```
<owl:Class rdf:ID="ISBN">
</owl:Class>
<owl:InverseFunctionalProperty rdf:ID="isbn">
 <rdfs:domain rdf:resource="#ISBN"/>
 <rdfs:range rdf:resource="#Book"/>
</owl:InverseFunctionalProperty>
```

等价类：

```
<owl:Class rdf:ID="Writer">
 <owl:equivalentClass rdf:resource="#Author"/>
</owl:Class>
...
<owl:Class rdf:ID="FictionBook">
 <owl:equivalentClass>
```

等价属性：

```
<owl:ObjectProperty rdf:ID="authorOf">
 <owl:equivalentProperty rdf:resource="#writerOf"/>
</owl:ObjectProperty>
```

同一性：

```
<Author rdf:ID="Joanne Rowling">
 <owl:sameAs rdf:resource="#J K Rowling" />
</Author>
```

（4）OWL-S 是一种基于 OWL 的 Web 服务本体，建立了 Services 的概念，它由 Profiles、Models（或 Processes）和 Groundings 组成。Service 使程序员能够根据服务自身的 URI 和参数，OWL 本体中需要映射到 Web 服务元素的信息，以及将本体信息转换到 Web 服务输出后，将 Web 服务输出转换回到本体元素的映射方法来指定 Web 服务。

① Service。一个 Service 表示一个 Profile。如下程序片段是 OWL-S 表示的 Amazon.com 电子商务服务。例如，下面的程序片段：

```
<service:Service rdf:ID="AmazonBookFinderService">
 <service:presents rdf:resource="#AmazonBookFinderProfile"/>
 <service:describedBy rdf:resource="#AmazonBookFinderProcess"/>
 <service:supports rdf:resource="#AmazonBookFinderGrounding"/>
</service:Service>
```

② Profile。Profile 基本上是一种描述性的结构。Profile 包括关于服务的名称和描述的信息，并给出了服务的输入输出参数，例如，下面的程序片段：

```
<profile:Profile rdf:ID="AmazonBookFinderProfile">
 <service:presentedBy rdf:resource="#AmazonBookFinderService"/>
 <profile:serviceName xml:lang="en">Amazon Book
```

```
 Finder</profile:serviceName>
 <profile:textDescription xml:lang="en">This service returns
 the information of a book whose title best matches the
 given string, from the Amazon E-Commerce
 Service.</profile:textDescription>
 <profile:hasInput rdf:resource="#BookName"/>
 <profile:hasOutput rdf:resource="#ItemInfo"/>
</profile:Profile>
```

③ Process。Service 由 Model 描述，后者采用某种 Process 的形式。Process 可能非常基础（就是说一个 AtomicProcess 仅执行一个事务），也可能很复杂（CompositeProcess），例如，下面是一个 AtomicProcess：

```
<process:AtomicProcess rdf:ID="AmazonBookFinderProcess">
 <service:describes rdf:resource="#AmazonBookFinderService"/>
 <process:hasInput rdf:resource="#BookName"/>
 <process:hasOutput rdf:resource="#ItemInfo"/>
</process:AtomicProcess>
```

④ 输入和输出参数，例如下面的程序片段：

```
<process:Input rdf:ID="BookName">
 <process:parameterType rdf:datatype=
 "&xsd;#anyURI">&xsd;#string</process:parameterType>
 <rdfs:label>Book Name</rdfs:label>
</process:Input>
<process:Output rdf:ID="ItemInfo">
 <process:parameterType rdf:datatype=
 "&store;#StockItem</process:parameterType>
 <rdfs:label>Item Info</rdfs:label>
</process:Output>
```

⑤ Grounding。Service 中的第三个也是最复杂的一个元素就是 Grounding。Service 支持 Grounding。简单地说，Grounding 将过程链接到一个消息映射来实现和 Web 服务的通信。在当前版本中，OWL-S 仅支持 WSDL grounding，但从原理上来说也能支持 REST grounding。OWL 中的 WSDL grounding 指定了服务的 URL、端口类型和目标操作、输入消息、输入消息映射列表（将输入参数和 Web 服务中的 WSDL 消息部分联系起来）、输出参数和输出消息映射列表。输出映射可以包括 XSLT 转换的使用，通常每个不同的 Web 服务通常都需要一个 Service 对象（包括 Profile、Process 和 Grounding），例如下程序片段：

```
<grounding:WsdlAtomicProcessGrounding
 rdf:ID="AmazonBookFinderProcessGrounding">
 <grounding:owlsProcess rdf:resource="#AmazonBookFinderProcess"/>
 <grounding:wsdlDocument rdf:datatype=
"&xsd;#anyURI">http://webservices.amazon.com/AWSECommerceService/AWSE
CommerceService.wsdl</grounding:wsdlDocument>
```

```
 <grounding:wsdlOperation>
 <grounding:WsdlOperationRef>
 <grounding:portType rdf:datatype=
"&xsd;#anyURI">http://webservices.amazon.com/AWSECommerceService/2006
-06-28</grounding:portType>
 <grounding:operation rdf:datatype=
"&xsd;#anyURI">http://webservices.amazon.com/AWSECommerceService/Item
Search</grounding:operation>
 </grounding:WsdlOperationRef>
 </grounding:wsdlOperation>
 <grounding:wsdlInputMessage rdf:datatype=
"&xsd;#anyURI">http://webservices.amazon.com/AWSECommerceService/Item
SearchRequest</grounding:wsdlInputMessage>
 <grounding:wsdlInput>
 <grounding:WsdlInputMessageMap>
 <grounding:owlsParameter rdf:resource="#BookName"/>
 <grounding:wsdlMessagePart rdf:datatype=
"&xsd;#anyURI">http://webservices.amazon.com/AWSECommerceService/Titl
e</grounding:wsdlMessagePart>
 </grounding:WsdlInputMessageMap>
 </grounding:wsdlInput>
 <grounding:wsdlOutputMessage
 rdf:datatype="&xsd;#anyURI">http://webservices.amazon.com/AWSECommerceSe
rvice/ItemSearchResponse</grounding:wsdlOutputMessage>
 <grounding:wsdlOutput>
 <grounding:WsdlOutputMessageMap>
 <grounding:owlsParameter rdf:resource="#StockInfo"/>
 <grounding:wsdlMessagePart
rdf:datatype="&xsd;#anyURI">http://webservices.amazon.com/AWSECommerceServ
ice/ItemSearchResponse</grounding:wsdlMessagePart>
 <grounding:xsltTransformationString>
 <![CDATA[
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
 <xsl:output method="xml" version="1.0" encoding="UTF-8"
indent="yes"/>
 <xsl:template match="/ ">
 <rdf:RDF
 xmlns:rdf "http://www.w3.org/1999/02/22 rdf syntax ns#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 <store:StockItem>
 <store:itemPrice>
 <xsl:value-of
select "http://webservices.amazon.com/AWSECommerceService/ItemSearchResponse/
OfferSummary/LowestPrice"/>
```

```
    </store:itemPrice>
    </store:StockItem>
    </rdf:RDF>
    </xsl:template>
   </xsl:stylesheet> ]]>
    </grounding:xsltTransformationString>
    </grounding:WsdlOutputMessageMap>
    </grounding:wsdlOutput>
   </grounding:WsdlAtomicProcessGrounding>
```

5.7.3 WSMO

WSMO 是欧洲 ESSI（European Telecommunications Standards Institute）研究语义 Web 服务的组织，该组织推出了 WSMO（Web Service Modeling Ontology）、WSML（Web Service Modeling Language）和 WSMX（Web Service Modeling eXecution environment）三个语义 Web 服务标准，其中 WSMO 主要由 Goals、Web Service、Mediations 和 Ontology 四部分组成。其中 Goals 表示一个客户期望通过 Web 服务完成一个具体的目标，它是一个请求者和提供者的松散耦合的目标本体，由 OO 和 GG 中介器约束；Mediations 表示为处理不同的本体实例而用中介器辅助设备进行组件间连接，它是解决在数据、过程以及协议层上不兼容问题的核心概念，即中介器级别包括：数据层（数据源）、协议层（传递模式）、过程层（业务流程）。其目的是通过为语义 Web 服务的核心元素提供本体化说明，从而更好地支持 Web 服务的发现、整合及交互。图 5-42 所示是 WSMO 组成结构。

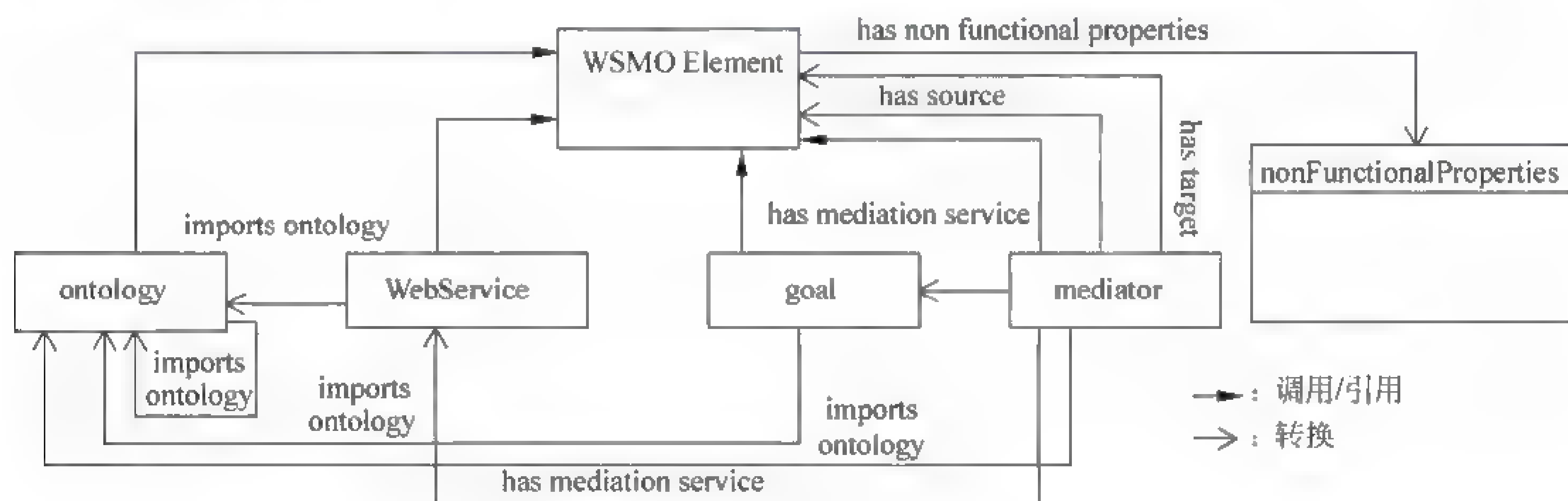

图 5-42　WSMO 组成结构

同时，WSMO 中介器在语义 Web 服务领域提出了一个全新的概念，图 5-43 是中介器组成结构，它是 WSMO 对语义 Web 服务的重要贡献之一。其通过对服务接口描述来实现服务组合的，以及通过本体实例完成所有数据交换，而服务接口（Service Interfaces）又是从编排（Choreography）和排列（Orchestration）两个角度实现的，编排是指其他 Web 服务的控制结构和交互功能集合，排列是提供 Web 服务的消费接口。同时，在进行服务功能描述仍使用了 WSDL，而本书使用了基于本体的语义识别服务功能描述，以增强 WSDL 语义。使用时可用如下方法来描述服务接口：

（1）Vocabulary(Ω)：服务本体的模式用服务接口 *i* 描述，交互信息用 *mutualinfor*=<*in*, *out*, *shared*, *controlled*>描述。

（2）States($\omega(\Omega)$)：在信息空间中描述一个稳定的状态，并通过本体实例的值定义属性。

（3）Guarded Transition($GT(\varphi)$)：表示状态转换，为区别于编排和排列之间的不同，额外的还有 *add*, *delete*, *update* 的结构，表示结构为 *if* (*condition*) *then* (*action*)。

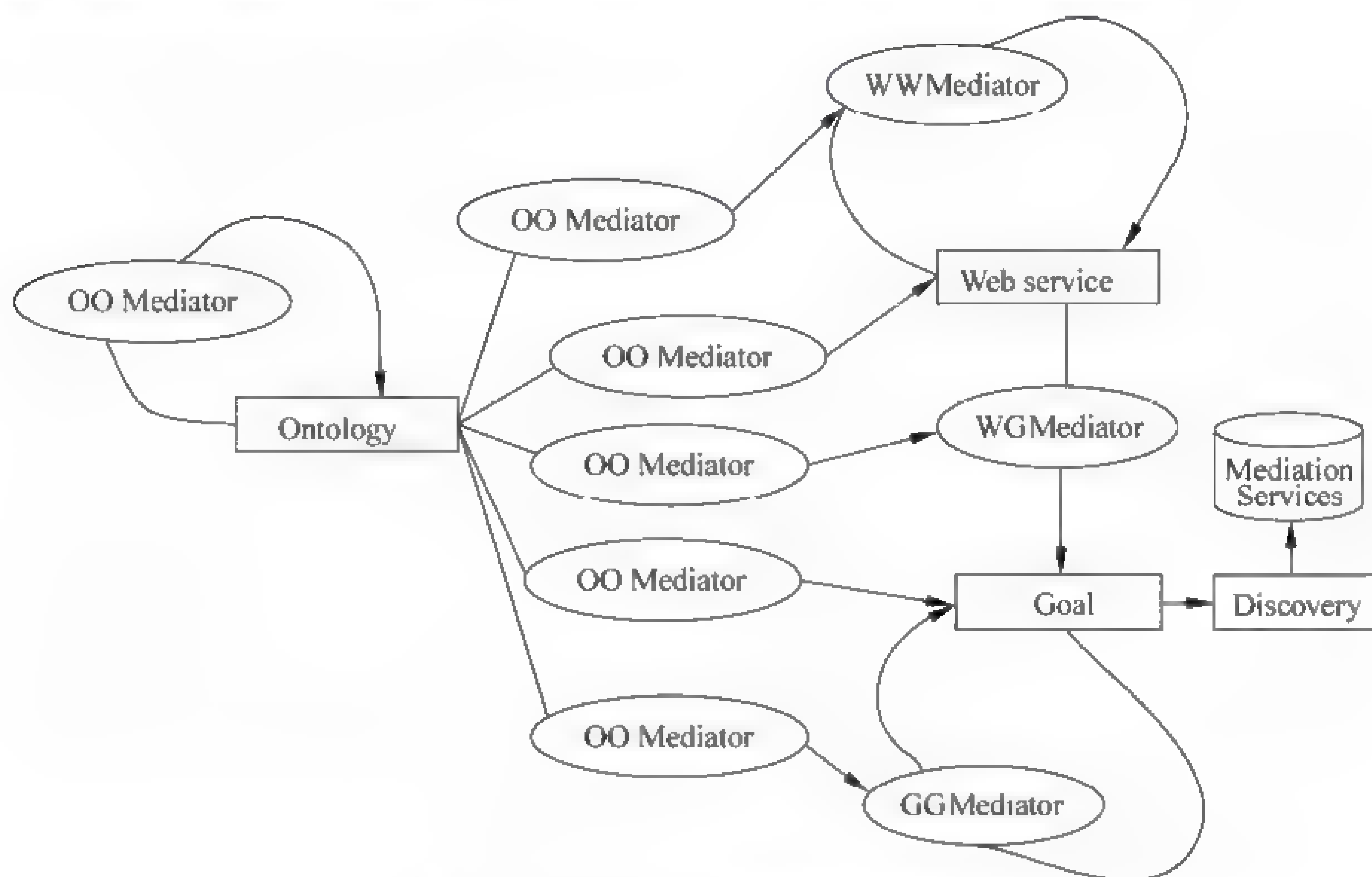

图 5-43　WSMO 中介器结构联系图

在 WSMO 中进行服务发现时，一般可有关键字、控制谓词、语义三种发现方法，然后进行匹配，实现 Goals（G）和 WebService（WS）通信，如图 5-44 所示。

图 5-44　服务发现匹配示意图

定义 1　一个服务的输出集合由 $ws(x)$来定义，公式为：

$$W:\forall x.(\varphi(x)\leftrightarrow ws(x))$$

式中，$\varphi(x)$表示一个任意用至多一个变量 x 优先处理的公式，φ表示与本体相关，

同样，一个目标对象由 $g(x)$来定义，公式为：

$$G:\forall x.(\phi(x)\leftrightarrow g(x))$$

式中，$\phi(x)$表示一个任意用至多一个变量 x 优先处理的公式，ϕ表示一定与本体相关，因此定义如下的服务发现匹配方法：

精确匹配：　$G,WS,O,M \models \forall x.(G(x\leftrightarrow WS(x))$

嵌入匹配：　$G,WS,O,M \models \forall x.(G(x)\rightarrow WS(x))$

$$G,WS,O,M \models \forall x.(WS(x)\leftarrow G(x))$$

包含匹配： $G,WS,O,M \models \forall x.(G(x) \leftarrow WS(x))$

$G,WS,O,M \models \forall x.(WS(x) \rightarrow G(x)$

相交匹配： $G,WS,O,M \models \exists x.(G(x) \wedge WS(x))$

没有匹配： $G,WS,O,M \models \neg x.(G(x) \wedge WS(x))$

式中：O 是表示本体，M 是匹配标志，x 表示参与匹配的本体。

例 1 服务组合集成的信息一致性和通信兼容性操作方法

（1）信息一致性（informationCompatibility）：

Step1　要是服务编排已适合Ω，则定义

$$\Omega_{in}(WSc_1) \text{ and } \Omega_{shared}(WSc_1) = \Omega_{out}(WSc_2) \text{ and } \Omega_{shared}(WSc_2)$$

Step2　从服务发现中定义服务组合匹配

$$OntWSc_{s1}, OntWSc_{s2}, O, M \models \exists x.\ (\Omega_{s1(in \cup shared)}(x) \wedge \Omega_{s2(out \cup shared)}(x))$$

Step3　若服务编排使用同种本体，此时服务接口处选择 OO 中介器，表示为：

$$SemanticInteroperability(S_1, S_2, ..., S_n)$$

（2）通信兼容性：

Step1　可编排状态

$$\omega_x(C(S1, S2)) \text{ if } informationCompatibility\ (\Omega S1(\omega_x), \Omega S2(\omega_x))$$

表示 $S1$ 中的 GT 转换到 $\omega_x(S1)$时，满足 $S2$ 中的 GT 转换到 $\omega_x(S2)$的条件，其中 C 表示混合两个服务本体（下同）；

Step2　开始状态

$$\omega_\emptyset(C(S1, S2)) \text{ if } \Omega S1(\omega_\emptyset)=\emptyset \text{ and } \Omega S2(\omega_\emptyset)=\emptyset \text{ and } \exists\ \omega_1(C(S1, S2))$$

表示最初状态没有本体编排参与者，也没有信息发生时，可用服务编排状态开始相互作用；

Step3　终止状态

$$\omega_T(C(S1, S2)) \text{ if } \Omega S1(\omega_T)=noAction \text{ and } \Omega S2(\omega_T)=noAction \text{ and } \exists \omega_T(C(S1, S2))$$

表示存在服务编排参与者终止状态，并有一个可用服务编排序列继续实行服务组合，直到完成相应的业务流程需求。

（3）OO 中介器：用不同的请求输入本体，使之匹配，保证语义互用。

例 2　OO 中介器使用方法

```
Class ooMediator sub-Class mediator
 hasSource type {ontology, ooMediator}
...
Ωin: Ontology S1
     Ontology S2
...
Mediator OO Mediator, Goal
merge s1, s2 and s1 subclassof s2
discovery Mediation Services
Ωout Ontology S3
```

WW 中介器：不同的服务使之能互用，且支持服务间自动协作匹配。

GG 中介器：连接目标和 Web 服务，允许定义目标本体，支持现有目标重用，通过 OO

中介器解决本体间可能的错匹配。

WG 中介器：连接 Web 服务到目标，并且解决可能的错匹配。

例 3　服务接口连接编排与排列实现服务组合集成直接应用方法（黑体表示 WSMO 中的关键词）：

```
Vocabulary:
Concept A in Ωin
Concept B in Ωout
Ωin hasValues {
  concept A [att1 ofType X att2 ofType Y]
...}
Ωout hasValues {
  concept B [att1 ofType P att2 ofType Q]
...}
State ω1 //接收本体实例 a
a memberOf A [att1 hasValue x att2 hasValue y]
        Guarded Transition GT(φ1)
            IF (a memberOf A [att1 hasValue x ])
THEN (b memberOf B [att2 hasValue m ])
      State ω2 //发送本体实例 b
            a memberOf A [att1 hasValue x,att2 hasValue y]
b memberOf B [att2 hasValue m]
```

目前，实现 WSMO 的环境是 WSMX，如图 5-45 所示；且定义了三类实体入口点，如图 5-46 所示。本书用一个网上购书系统来应用举例，它的服务发现通过目标和 Web 服务（capability）来实现；服务组合（消费）是通过服务接口（编排+排列）来完成；调用方法用 Grounding（WSDL/SOAP，基于本体）来完成，使用 Ontology 来完成服务功能的描述；异构处理用中介器来完成，具体实现步骤是：

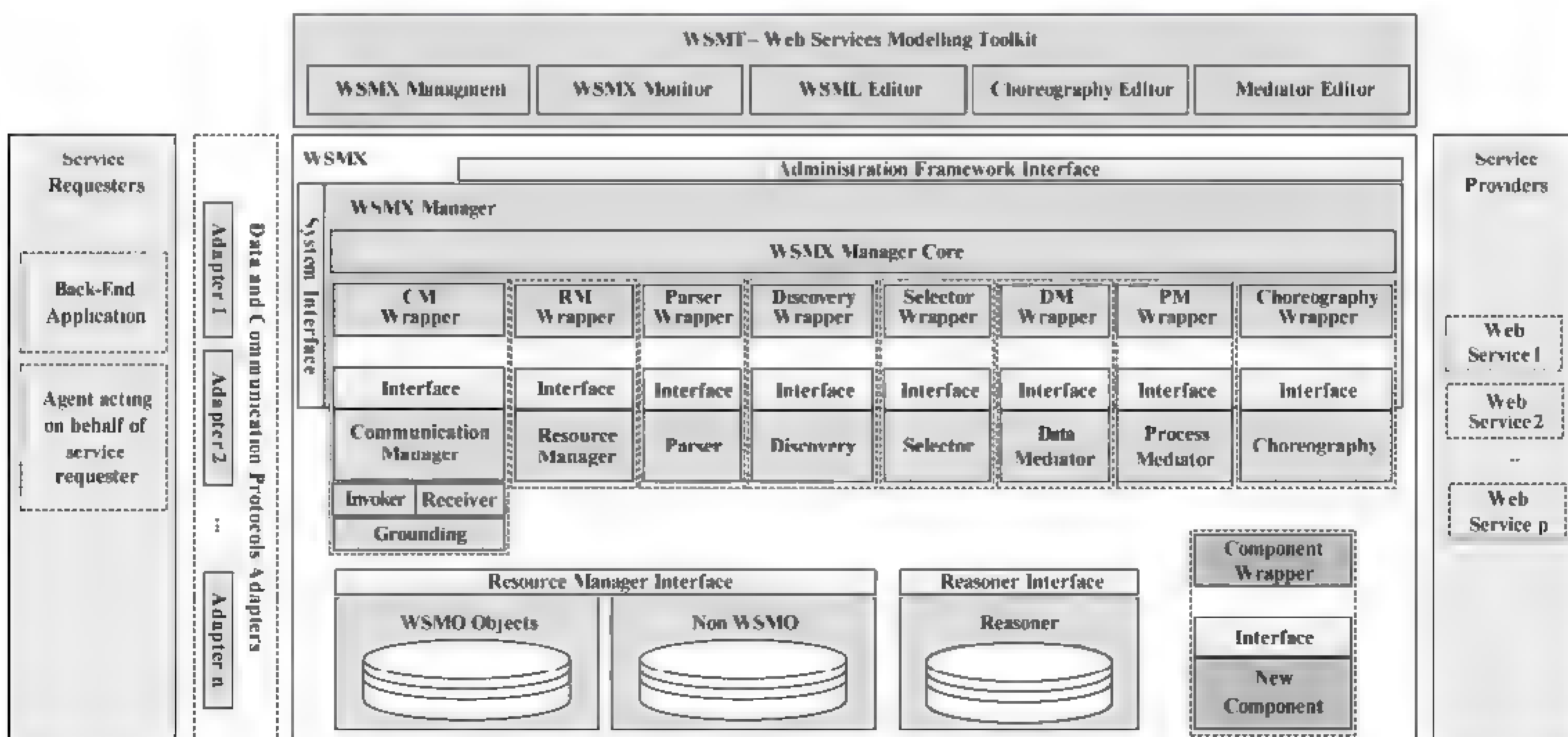

图 5-45　WSMX 体系结构图

（1）进行基于本体的目标描述，

（2）服务描述，如下程序列表是描述服务的方法，

（3）中介器定义，

（4）服务接口编排或定义。

① storeEntity(WSMOEntity)：Confirmation 入口点，其中 Confirmation 表示发送或接收的确认信息（下同），它表示为存储一些 WSMO 关系实体（Web Services, Goals, Ontologies）提供一种管理接口；

② realizeGoal(Goal, OntologyInstance)：Confirmation：服务请求期望 WSMX 去发现和调用没有额外的 Web 服务消息；

③ receiveGoal(Goal, OntologyInstance, Preferences):WebService[]：为给定的目标创建服务列表，并且使服务请求能指定许多服务被返回；

④ receiveMessage(OntologyInstance,WebServiceID, ChoreographyID):ChoreographyID：上下文的会话为调用提供所有必要的数据，且在服务接口间调用服务编排的执行和中介器处理。

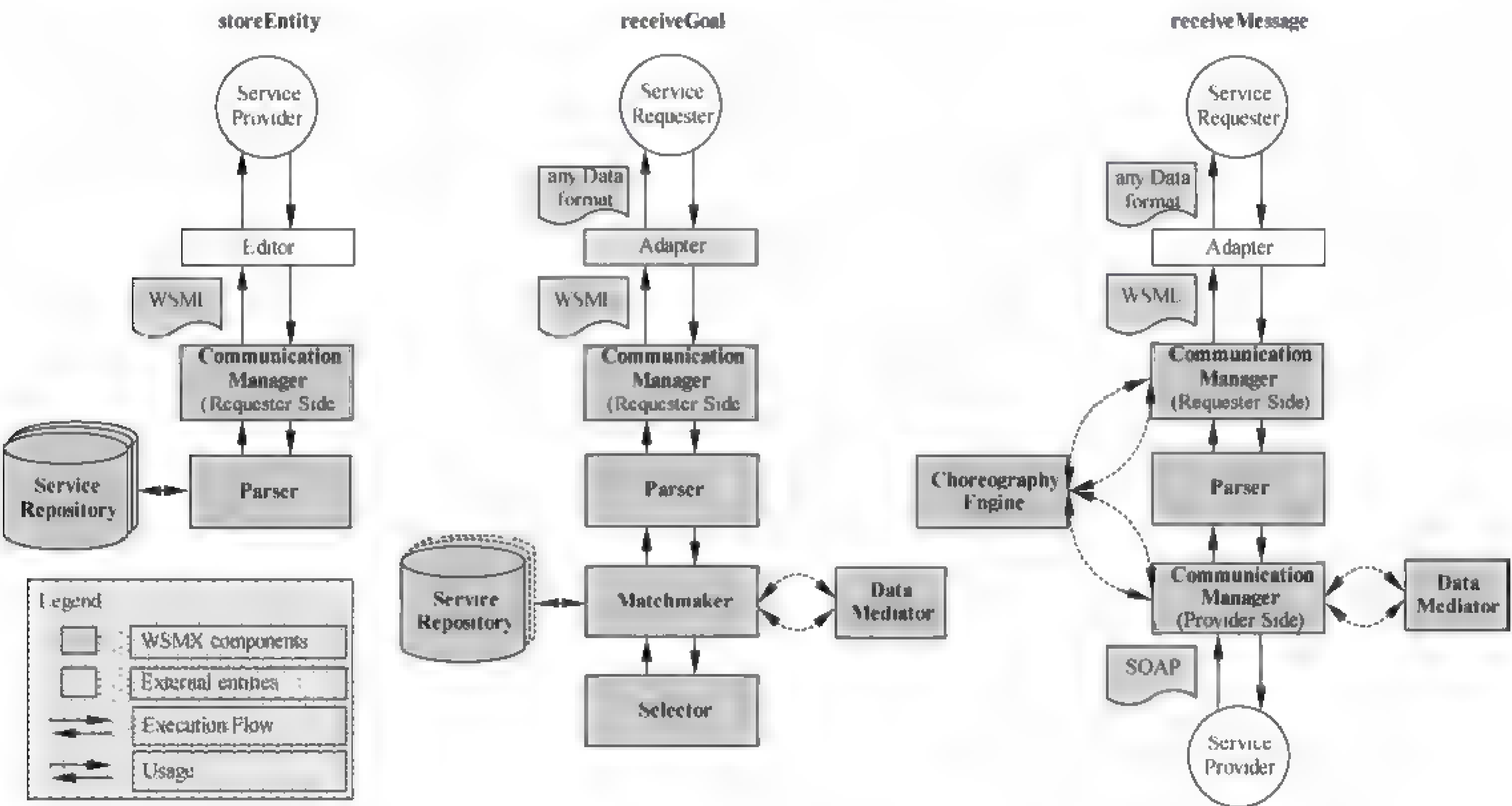

图 5-46　WSMX 实体入口点图

多用户通过登录具有 GBSS 的电子商务网站订购图书时，登录时可能会来自不同的安全服务、登录日志服务、信誉等级服务、检查信用卡帐户余额服务等；接着通过合适的、最优的服务组合去执行登录任务后，这时不同的用户可能同时需要电子商铺请求订购不同的图书，比如他们可能同时需要买图书 A、图书 B、图书 C，如图 5-47 所示。其基本信息描述举例见程序清单 5-35，服务编排举例见程序清单 5-36。

程序列表 5-35：

```
capability BuyEBcapability
 sharedVariables {?creditCard, ?initialBalance, ?item, ?user}
 precondition
```

```
  definedBy
   ?reservationRequest[
        reservationItem hasValue ?item,
        user hasValue ?user,
        payment hasValue ?creditcard,
      ] memberOf gds#bookRequest and
     ((?item memberOf gds# BookA) or (?item memberOf gds# BookB))
or (?item memberOf gds# BookC)) and
     ?creditCard[balance hasValue ?initialBalance] memberOf po#creditCard
 assumption
  definedBy
   po#validCreditCard(?creditCard) and
   (?creditCard[type hasValue po#visa] or ?creditCard[type hasValue
   po#mastercard])
```

程序列表 5-36：

```
choreography Large-EBIESBehaviorInterface
 importsOntology { "http://Hostlocal/ontologies/Large-BOOKSOntology", …}
 vocabularyIn {reservationRequest, …}
 vocabularyOut {reservation, …}
guardedTransitions Large-BOOKBehaviorInterfaceTransitionRules
if (...)
...
then
...
```

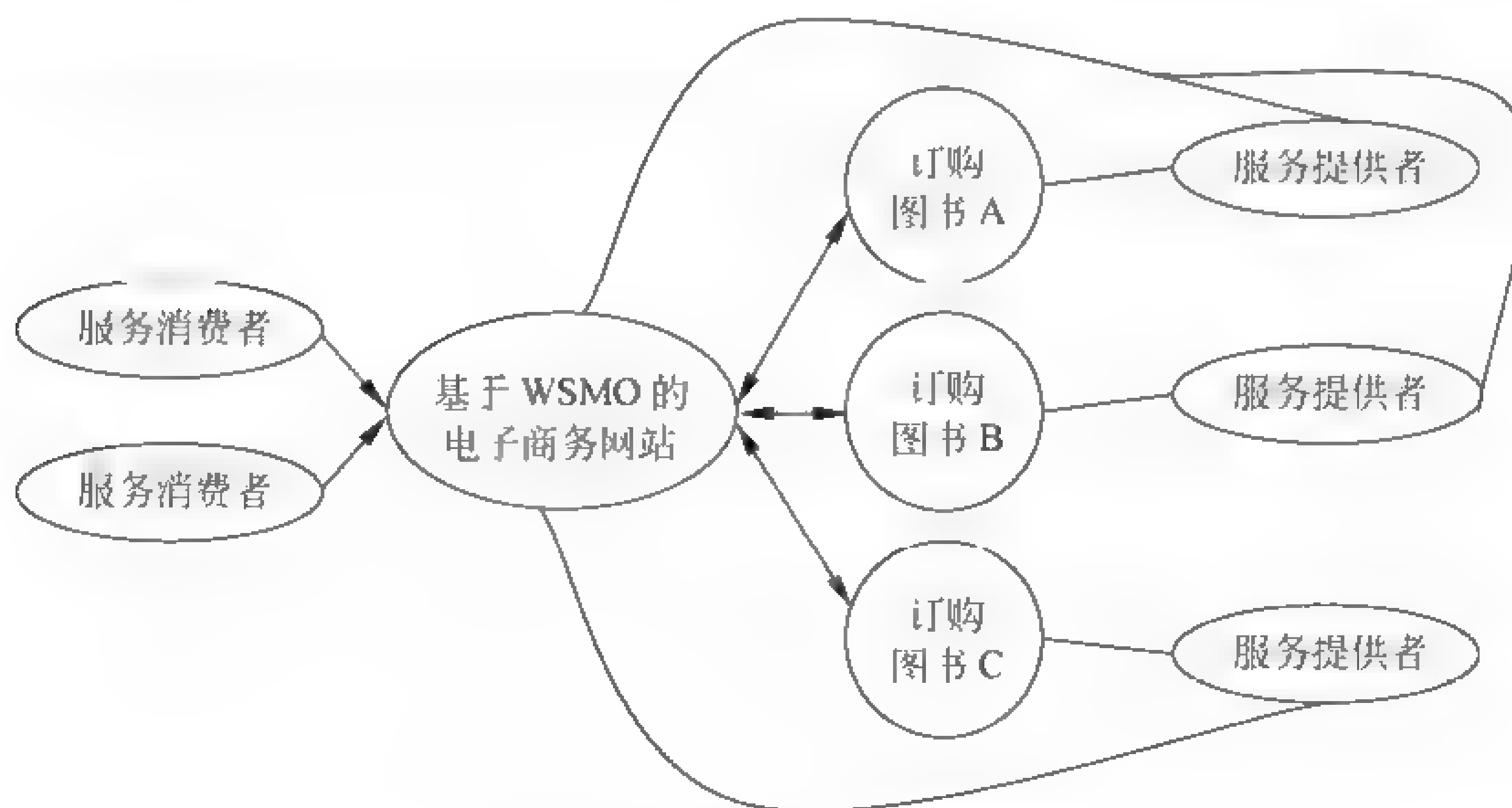

图 5-47　用户购书表示图

然而，OWL-S 与 WSMO 也存在重要的区别：OWL-S 的基础主要为描述逻辑，WSMO 主要是框架描述，即在服务聚合时，OWL-S 使用流程模型，WSMO 使用中介器，主要区别如下：

（1）OWL-S 是一种描述 Web 服务的本体语言，目的是丰富现有的 Web 服务标准，并提

供概念框架和相关的推理机制去描述 Web 服务域，且使本体是 Web 服务间互操作的基本元素，用 Profiles 表示现有和需求的性能。WSMO 是一种语义 Web 服务核心元素的概念模型，包括：本体、Web 服务、目标和中介器；中介器做一个关键元素，Web 服务以服务接口的编排和排列进行推理，本体作为数据模型，且每个资源基于本体描述，每个数据元素交换是一个本体实例。OWL-S Profiles 相似于 WSMO 性能、性能和非功能性属性。

（2）OWL-S 与 WSMO 在本体需求方面都采用一种类似的观点、方法和语义，但是信赖不同的逻辑：OWL 基于 OWL/SWRL，描述分类知识，且 SWRL 提供接口规则，同时采用 FLOWS 方式。而 WSMO 是基于 WSML 在逻辑描述和逻辑语言指导中的一系列语言，且用一种基本的性能和扩展实现描述。

（3）OWL-S 模型相似于 WSMO 服务接口，OWL-S 通过 Web 服务过程处理模型描述业务流程，包括消息和集合的概念，且处理模型被 SWRL/FLOWS 延伸。而此时 WSMO 通过服务接口分离编排和排列。

（4）OWL-S 正式语义已经在不同的框架实现了开发，如 SituationCalculus、Petri、Nets 等。

（5）OWL-S 提供默认规则映射到 WSDL，在 Web 服务描述与接口实现间清晰分离。而 WSMO 也定义了一种规则映射到 WSDL，但是目标是基于本体 Grounding，这样避免通过服务用例损失本体描述。

5.8 数据库访问技术

在当前的企业、信息化建设解决方案中，都离不开数据的获取，而这些数据往往存储在数据库中。然而，在软件构架中，通常采用的三层架构（或是它的变形 - N 层架构）常常是架构师们的首选，这当中包括表现层，商业逻辑层和数据访问层。而其中数据访问层，它负责和应用中的各种数据源打交道，诸如 DB2、Oracle 等关系型数据源、XML 数据，以及其他种类的非关系型数据、Web 服务、各种特别的遗留系统等；最后将它们整合起来，为商业逻辑层提供统一的数据服务。而对于数据持久开源软件框架 Hibernate，它内嵌了对 C3P0、Proxool、JNDI 数据源等数据库连接池的支持，这为 Hibernate 访问多种数据库提供了功能上的支持。这是因为 Hibernate 作为一个强大的数据持久层组件，它在实现数据库连接方面的扩展性也是非常强大的。

从数据库技术诞生到现在，已经有了数十年的历史；相应的，从最早的读写磁盘文件开始，数据访问技术也不断的推陈出新，变得越来越丰富。下面总结描述 ODBC、JDBC、ADO.NET 和 pureXML 等主流数据库访问技术。

5.8.1 ODBC

ODBC（Open Database Connectivity，开放数据库互连）是由 Microsoft 公司基于 X/OPEN CLI（Common Language Infrastructure）提出的用于访问数据库的应用程序编程接口，主要完成应用程序和数据库系统之间的中间件功能。基于 ODBC 的应用程序通过 ODBC 提供的 API 与数据库进行交互，在避免了应用程序直接操作数据库系统的同时，极大的增强了应用

程序的可移植性、扩展性和可维护性。

ODBC 提供了一种标准的 API 方法来访问 DBMS（Database Management System）。这些 API 利用 SQL 来完成其大部分任务。ODBC 本身也提供了对 SQL 语言的支持，用户可以直接将 SQL 语句送给 ODBC。ODBC 的设计者们努力使它具有最大的独立性和开放性，即与具体的编程语言无关，与具体的数据库系统无关，与具体的操作系统无关。

ODBC 应用程序与 CLI 应用程序之间主要的不同在于装载数据库驱动程序的方式。在 ODBC 环境中，Driver Manager 为应用程序提供接口，且它还动态地为应用程序所连接到的数据库服务器装载必需的驱动程序。应用程序对 ODBC DriverManager 进行的每个 ODBC 函数调用都被转到适当的数据源驱动程序进行处理。在 CLI 环境中，应用程序将自己装载数据库驱动程序，因此只能与一种数据源驱动程序绑定在一起，如图 5-48 所示。

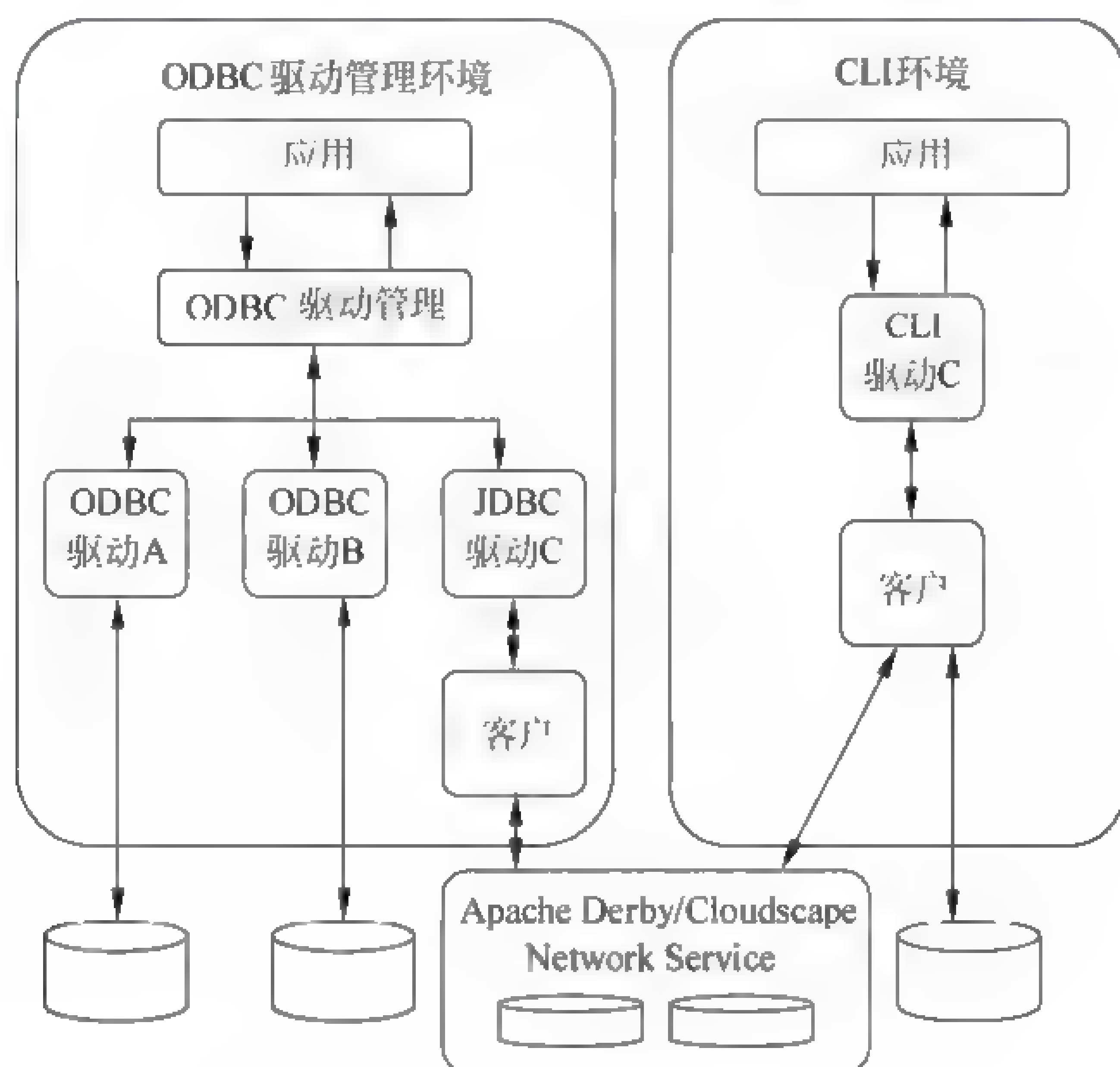

图 5-48　ODBC 与 CLI 结构

ODBC 的运用形态通常是由应用程序经过一个称为 ODBC 管理器的工具，间接调用 ODBC 驱动程序，从而访问对应的数据库。对于用户的应用程序而言，ODBC 驱动程序是相对不可见的。用户只需要在 ODBC 管理器中配置相应的数据库的数据源信息，并登录相应的 ODBC 驱动程序即可，图 5-49 所示是 ODBC 体系结构。当前各个数据库厂商通常都为自己的数据库实现了 ODBC 驱动程序。从 Oracle、DB2、SQL Server 到微软的 Access 数据库，都实现了面向各自数据库产品的数据驱动程序。

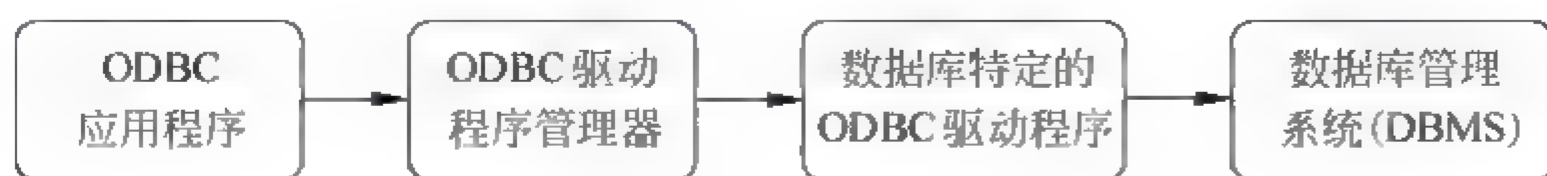

图 5-49　OBDC 体系结构

5.8.2 JDBC

Java 数据库连接（Java Database Connectivity，JDBC）是 Java 语言中用来规范客户端程序如何来访问数据库的应用程序接口，提供了诸如查询和更新数据库中数据的方法。JDBC 也是 Sun Microsystems 的商标，且 JDBC 是面向关系型数据库的。Microsoft 的开放式数据 ODBC 和 JDBC 都基于 X/Open SQL 调用层接口规范。目前微软程序也支持 JDBC。在 J2SE 中，提供了一个称为 JDBC-ODBC 桥（JDBC-ODBC Bridge）的 API。通过 ODBC，JDBC-ODBC 桥驱动程序可以访问所有支持 ODBC 的关系型数据库。与 JDBC API 不同的是，这个驱动程序并不是由 Java 代码而是由机器码（native code）编写，并且不是开放源代码的。

JDBC 包含四种类型的驱动程序：JDBC-ODBC 桥，本地 API 驱动，网络协议驱动，本地协议驱动。依照 JDBC 规范，这四种类型的 JDBC 驱动程序体系结构可描述如下：

（1）这类驱动程序将 JDBC API 作为到另一个数据访问 API 的映射来实现，如开放 ODBC。这类驱动程序通常依赖本机库，这就限制了其可移植性。JDBC-ODBC 桥驱动程序就是该驱动程序的最常见的例子，即为 type1。

（2）这类驱动程序部分用 JAVA 编程语言编写，部分用本机代码编写。并使用特定于所连接数据源的本机客户端库。同样，由于使用本机代码，所以其可移植性受到限制，即为 type2。

（3）这类驱动程序使用纯 JAVA 客户机，并使用独立于数据库的协议与中间件服务器通信，然后中间件服务器将客户机请求传给数据源。即为 type3。

（4）这类驱动程序是纯 JAVA，实现针对特定数据源的网络协议。并且客户机直接连接至数据源。目前，有关 JDBC 最新的工业规范是 JDBC 3.0[①]，图 5-50 所示是网络协议驱动的 JDBC 结构（即 type4）。

图 5-50 网络协议的驱动结构

图 5-51 所示是 JDBC 到数据库的通信结构。

JDBC 也是一种用于执行 SQL 语句的 Java API，由一组用 Java 语言编写的类和接口组成，能为 SQLserver、Oracle、DB2、MySQL 等数据库提供统一访问。JDBC API 主要位于 JDK 中的 java.sql 包中（之后扩展的内容位于 javax.sql 包中）。主要包括如图 5-52 所示的功能，它们之间的关系如表 5-9 所示。

---

① http://www.jcp.org/en/jsr/detail?id=54 JSR54

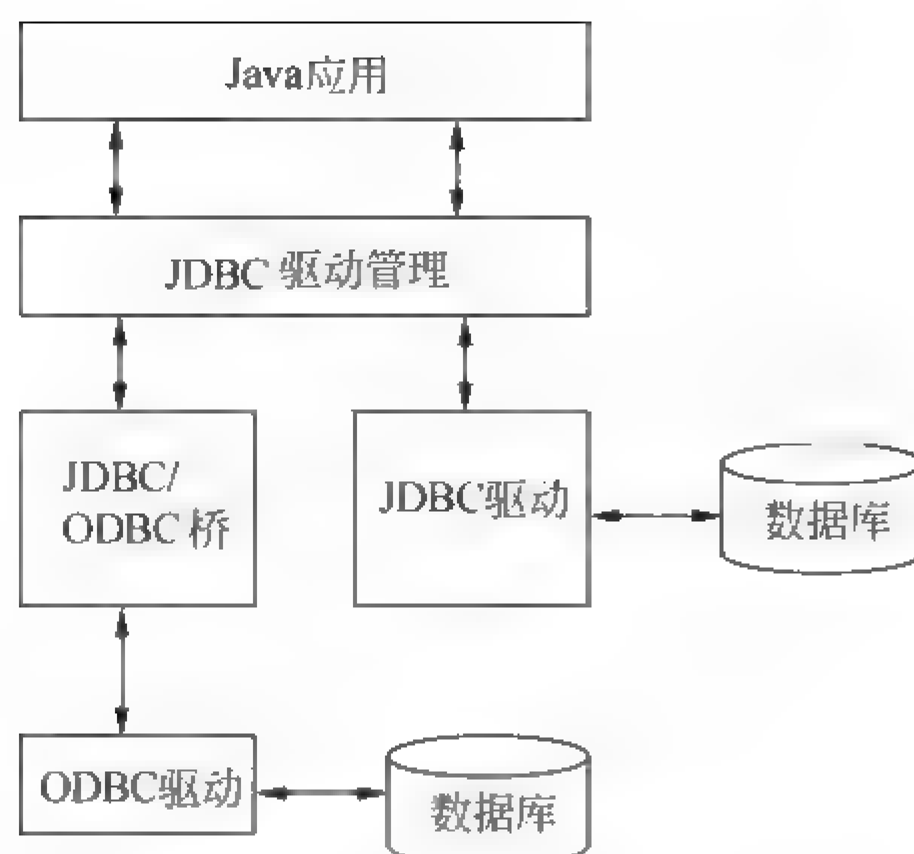

图 5-51 JDBC 到数据库的通信结构

表 5-9 JDBC 主要包括功能描述

功能项	描述
DriverManager	加载各种不同驱动程序（Driver），并向调用者返回相应的数据库连接
Driver	驱动程序，会将自身加载到 DriverManager 中去
Connection	数据库连接，负责与进行数据库间通信。可以产生用以执行 SQL 的 Statement
Statement	用以执行 SQL 查询和更新
PreparedStatement	用以执行包含动态参数的 SQL 查询和更新
CallableStatement	用以调用数据库中的存储过程
SQLException	代表在数据库连接的创建关闭和 SQL 语句的执行过程中发生的即错误

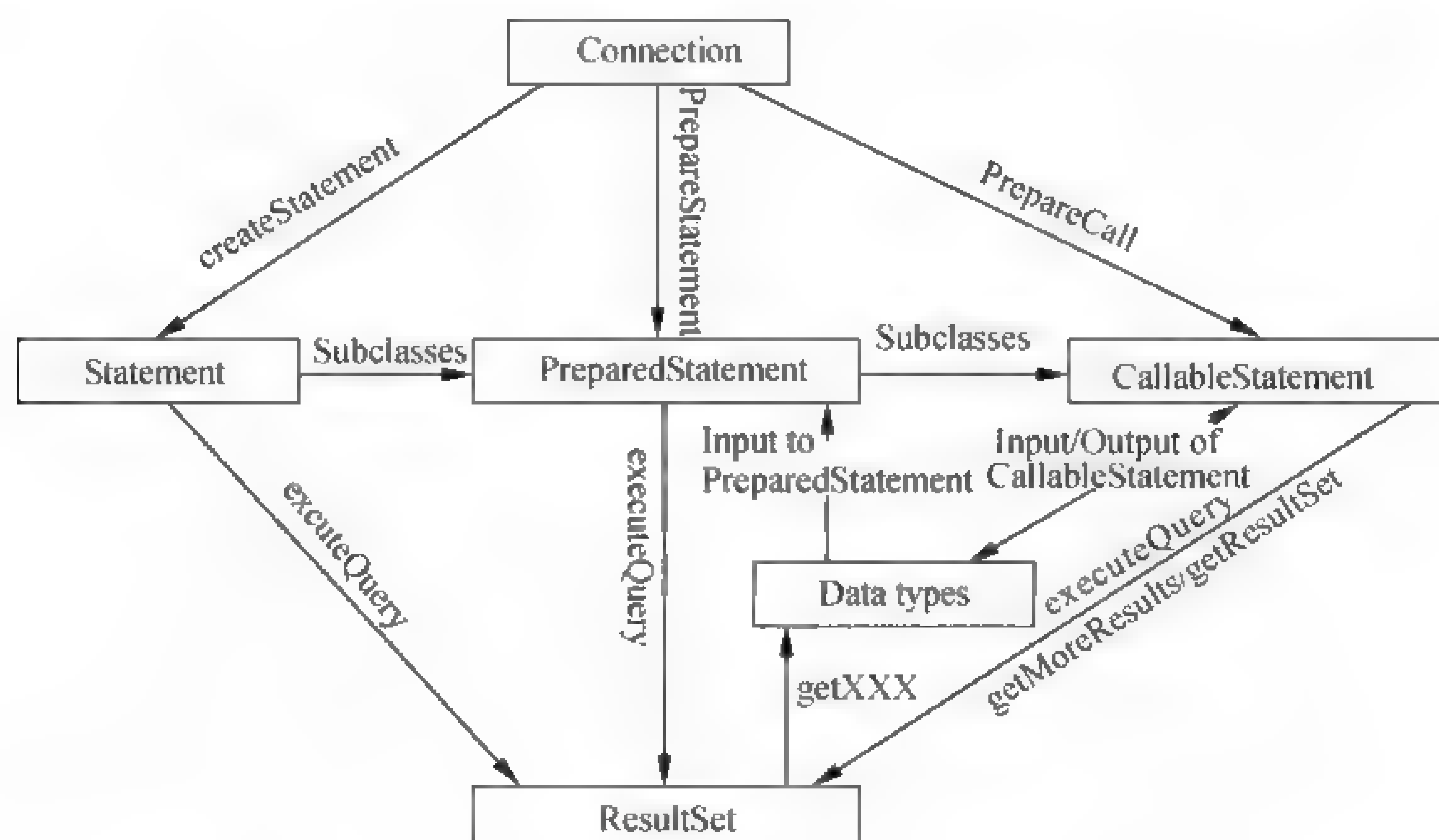

图 5-52 在 javax.sql 包中的 major classed 与接口间的关系

为有效提高数据库密集型应用程序的性能，通常采用连接池和语句池技术来实现，这是因为可以有效实现对象重用，而无须花费时间和资源重新创建对象。当应用程序与数据库频繁交互并且经常使用相同的参数重新建立连接时，通常使用连接池技术来完成；当应用程序

运行期间多次、重复执行 SQL 语句时，通常使用语句池技术。因此在 JDBC4.0 中完善了连接池技术，如图 5-53 所示。

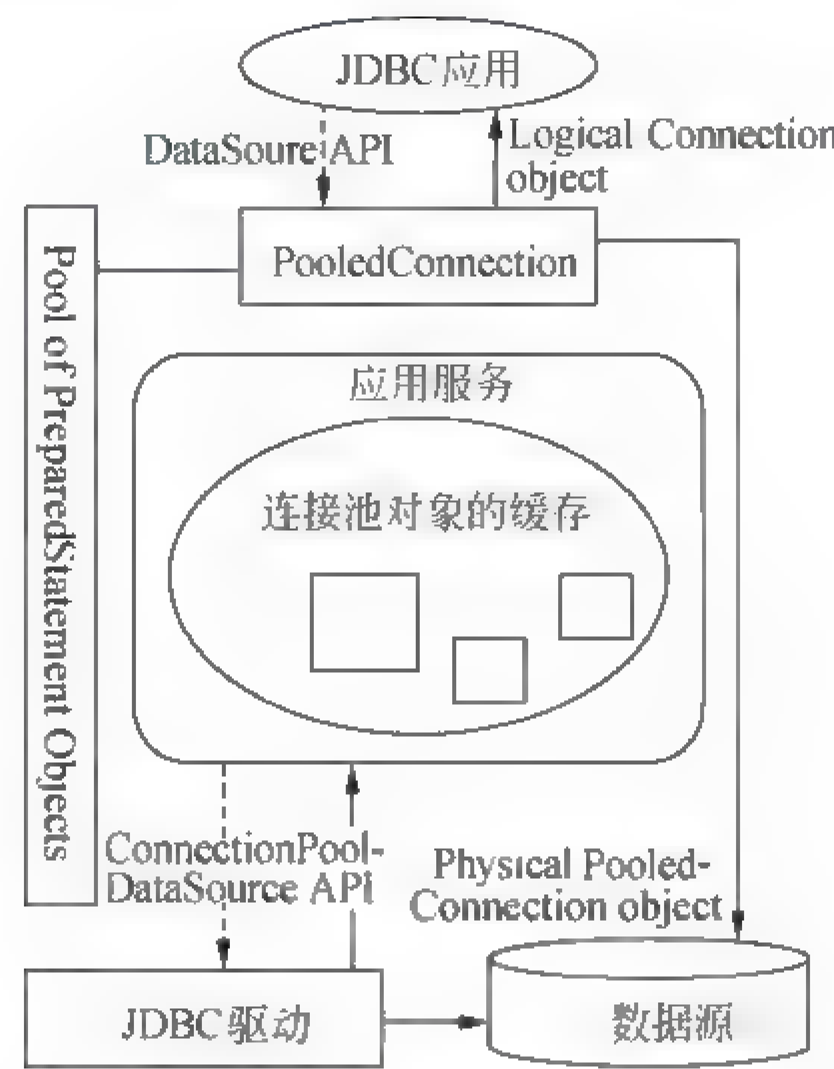

图 5-53　JDBC 连接池结构图

表 5-10 所示是基于 JDBC 数据库访问的开源软件列表。

表 5-10　基于 JDBC 的数据库访问技术的开源一览表

（来源：http://www.open-open.com/65.htm）

软件名	描述	网址
Excel JDBC Driver	一个可以把 Excel 中的数据导入数据库也可以反过来操作的 JDBC 驱动	http://sourceforge.net/projects/xlsql
C-JDBC	C-JDBC 作为开发源码的数据库群集中间件，可以让任何应用程序通过 JDBC 能够透明的访问数据库群集。数据库可以分布在多个结点并可以进行数据复制，C-JDBC 能够均衡在这些结点之间的查询负载。C-JDBC 是 GNU LGPL 许可证下的 ObjectWeb 项目	http://c-jdbc.ow2.org
RmiJdbc	基于 Java RMI 的 Client/Server JDBC Driver。所有 JDBC classes(例如 Connection, ResultSet 等）都被处理成分布式的 RMI 对象，因此可以远程访问任何支持 Jdbc API 的数据库。事实上 RmiJdbc 正是一座可以远程访问 JDBC Driver 的桥	http://rmijdbc.ow2.org
MM.MySQL	是一个 Java 开源（license：LGPL 与 GPL）的连接 MYSQL 数据库的 JDBC Driver	http://mmmysql.sourceforge. net
JTDS	一个 Microsoft SQL Server 数据库的 Jdbc driver	http://jtds.sourceforge.net

续表

软件名	描述	网址
xlSQL	xlSQL 是一个 Jdbc 驱动针对于 Excel（CSV, XML 和其他）文档数据源。利用 SQL 可以像操作数据库中的表格一样来读写文档。xlSQL 把 Excel 文件的目录映射成一个数据库，工作簿映射成 schemas，工作表映射成表格。查询文档可以使用 HSQL 或 MySQL 方言，但写入对象与数据需要用本地的(native)xlSQL	https://xlsql.dev.java.net http://xlsql.sourceforge.net
HA-JDBC	是一个 JDBC 驱动代理（proxy）它让其他 JDBC 驱动具备轻量的、透明化的群集处理能力	http://ha-jdbc.sourceforge.net
MS Jdbc Proxy	通过对 Microsoft JDBC Driver 进行包装以解决它运行速度慢及在处理 Object 和 Nvarchar 数据时的缺陷，使 MS Jdbc Driver 运行地更快和更好，同时使它能更好地支持 Hibernate 2.0 和 3.0	http://msjdbcproxy.sourceforge.net
LDBC	LDBC（Liberty DataBase Connectivity）是一个基于 ANSI-SQL 与 JDBC 标准的 JDBC 驱动器，它提供了一个与厂商无关的数据库访问。利用 LDBC 可以在不修改任何源代码的情况下就能让你的应用程序可运行在所有主流数据库上	http://ldbc.sourceforge.net
Virtual JDBC	VJDBC 是一个 JDBC type 3 驱动器。它提供一个用于通过各种不同的网络协议来远程访问 JDBC 数据源的 client-server 模型	http://vjdbc.sourceforge.net
CsvJdbc	提供了 Java 访问 csv 文件的的 JDBC 驱动，它其实是把一个 csv 文件当做一个数据库表来操作，提供简单的查询	http://csvjdbc.sourceforge.net
XLSJDBC	是一个只读 JDBC 驱动器提供了 Java 访问 XLS 文件的能力，它把一个 XLS 文件当做一个 SQL 数据库表来进行查询	http://xlsjdbc.sourceforge.net
WS-JDBC	是一个 client/server JDBC 驱动器，其中服务器部分是以 Web 服务的方式来实现。这意味着这个定制的 JDBC 驱动器客户端可以通过 Internet 来调用相应的服务	http://ws-jdbc.sourceforge.net
Sequoia	是一个能够为任何数据库提供群集，负载平衡和容错服务的中间件。Sequoia 是 C-JDBC 项目的扩展	http://code.google.com/p/tungsten-replicator

续表

软件名	描述	网址
SQLiteJDBC	一个 SQLite 数据库的 JDBC Driver。它构建在 SQLite 3.3.x C 语言 API 之上，支持大部分 JDBC 标准，除了 Java date/time 类，完全支持 UTF-16	http://www.zentus.com/sqlitejdbc
PostgreSQL JDBC Driver	PostgreSQL 数据库 JDBC Driver 采用纯 Java（Type IV）实现，允许 Java 程序使用标准，不依赖于数据库的 Java 代码连接到 PostgreSQL 数据库。这个 Driver 实现了全部 JDBC3 标准，此外还增加了一些针对 PostgreSQL 特有的扩展	http://jdbc.postgresql.org
StelsXML	是一个 XML JDBC 驱动，它可以在 XML 文件上执行 SQL 查询和其他 JDBC 操作，支持大部分 ANSI SQL'92 的关键字，支持 XPath 表达式来定义表格和字段，不需要任何中间数据转换和映射操作，支持 inner 和 outer table joins，支持 integer、float、string 和 date/time 等数据类型，支持合计，数字/字符转换和其他 SQL 函数，支持用户自定义 SQL 函数，具有平台无关性	http://www.csv-jdbc.com/stels_xml_jdbc.htm
JDBC-LDAP	JDBC-LDAP Bridge Driver 能够使用 SQL 和 JDBC 存取目录上的信息	http://www.openldap.org/jdbcldap
log4jdbc	log4jdbc 是一个 JDBC 驱动器，能够记录 SQL 日志和 SQL 执行时间等信息。它使用 SLF4J（Simple Logging Facade）作为日志系统。特性包括：支持 JDBC3 和 JDBC4；支持现有大部分 JDBC 驱动；易于配置（在大部分情况下，只须要改变驱动类名并在 jdbc url 前加上“jdbc:log4”，设置好日志输出级别）；能够自动把 SQL 变量值加到 SQL 输出日志中，改进易读性和方便调试；能够快速标识出应用程序中执行比较慢的 SQL 语句；能够生成 SQL 连接数信息帮助识别连接池/线程问题	http://code.google.com/p/log4jdbc
Drizzle-JDBC	Drizzle-JDBC 是 drizzle 数据库的一个 JDBC 驱动。Drizzle 基于 MySQL 6.0 的源代码，并针对云和网络应用程序进行了优化。现在他们已经从原来的代码中去除了许多功能	https://launchpad.net/drizzle-jdbc
JDBC-Redis	是用于操作 NoSQL 数据库 Redis 的 JDBC 驱动，但这个项目并没有实现完整的 JDBC 规范，因为 Redis 不是一个关系型数据库。但是 Java 开发人员可以采用熟悉的 JDBC 接口来访问 Redis 数据库	http://code.google.com/p/jdbc-redis

续表

软件名	描述	网址
ADBCJ	ADBCJ（Asynchronous Database Connectivity in Java）是一个异步数据库驱动程序的 API for Java。ADBCJ 类似 JDBC 的，因为它是一个基于 SQL 的数据库交互的 API。关键的区别是，ADBCJ 连接到数据库，执行 SQL 查询，启动和停止事务，并从数据库断开都是无阻塞。目前有一个 ADBCJ 调用 JDBC 的驱动程序，使用线程池来实现并发。还有一个 MySQL 和 PostgreSQL 的本地驱动。原生驱动程序都是建立在高性能的网络框架 MINA 的基础上。目前正在测试衡量性能和线程池之间的 MINA 的基础和实现资源利用的差异	http://code.google.com/p/adbcj
cegojdbc	是一个用于 Cego 数据库系统的 JDBC 驱动器。需要 Java1.5 以上的运行环境。Cego 是一个采用 C/C++开发的开源数据库，实现了一个支持事务和 SQL 查询语言的关系数据库，支持一般的数据库功能以及索引、视图、存储过程等	http://freshmeat.net/projects/cegojdbc

5.8.3　ADO.NET

ADO.NET 是微软开发的一种基于.NET 框架的数据库访问技术，它具有完全基于 XML 和离线的数据访问计算模型，它与早期的 ADO 主要具有以下区别：

（1）ADO 是以 Recordset 存储，而 ADO.NET 则以 DataSet 表示。Recordset 看起来更像单张数据表。如果让 Recordset 以多表的方式表示就必须在 SQL 中进行多表连接；而 DataSet 可以是多个表的集合。

ADO 的运作是一种在线方式，这意味着不论是浏览或更新数据都必须是实时的；而 ADO.NET 则使用离线方式，在访问数据的时候 ADO.NET 会导入并以 XML 格式维护数据的一份副本，ADO.NET 的数据库连接也只有在这段时间需要在线。

（2）由于 ADO 使用 COM（Component Object Model）技术，这就要求所使用的数据类型必须符合 COM 规范，而 ADO.NET 基于 XML 格式，数据类型更为丰富并且不需要再做 COM 编排导致的数据类型转换，从而提高了整体性能。图 5-54 所示是 ADO 与 ADO.NET 的结构。

（3）ADO.NET 不是简单对 ADO 扩展，它提供了对关系数据、XML 和应用程序数据的访问，对 Microsoft SQL Server 和 XML 等数据源，以及通过 OLE DB 和 XML 公开的数据源提供一致的访问；并通过 ADO.NET DataSet 对象处理和缓存数据。

ADO.NET 通过数据处理将数据访问分解为多个可以单独使用或一前一后使用的不连续组件。ADO.NET 包含用于连接到数据库、执行命令和检索结果的.NET Framework 数据提供程序，如图 5-55 所示。

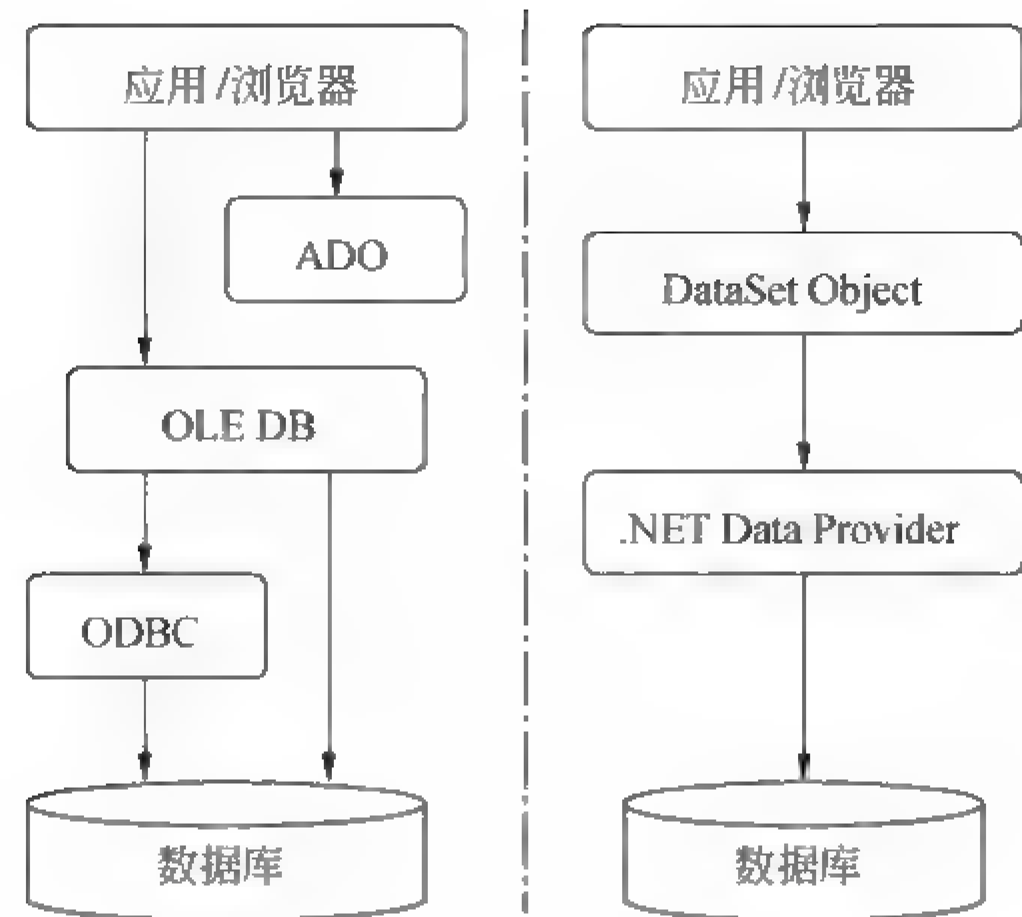

图 5-54　ADO 与 ADO.NET 结构比较

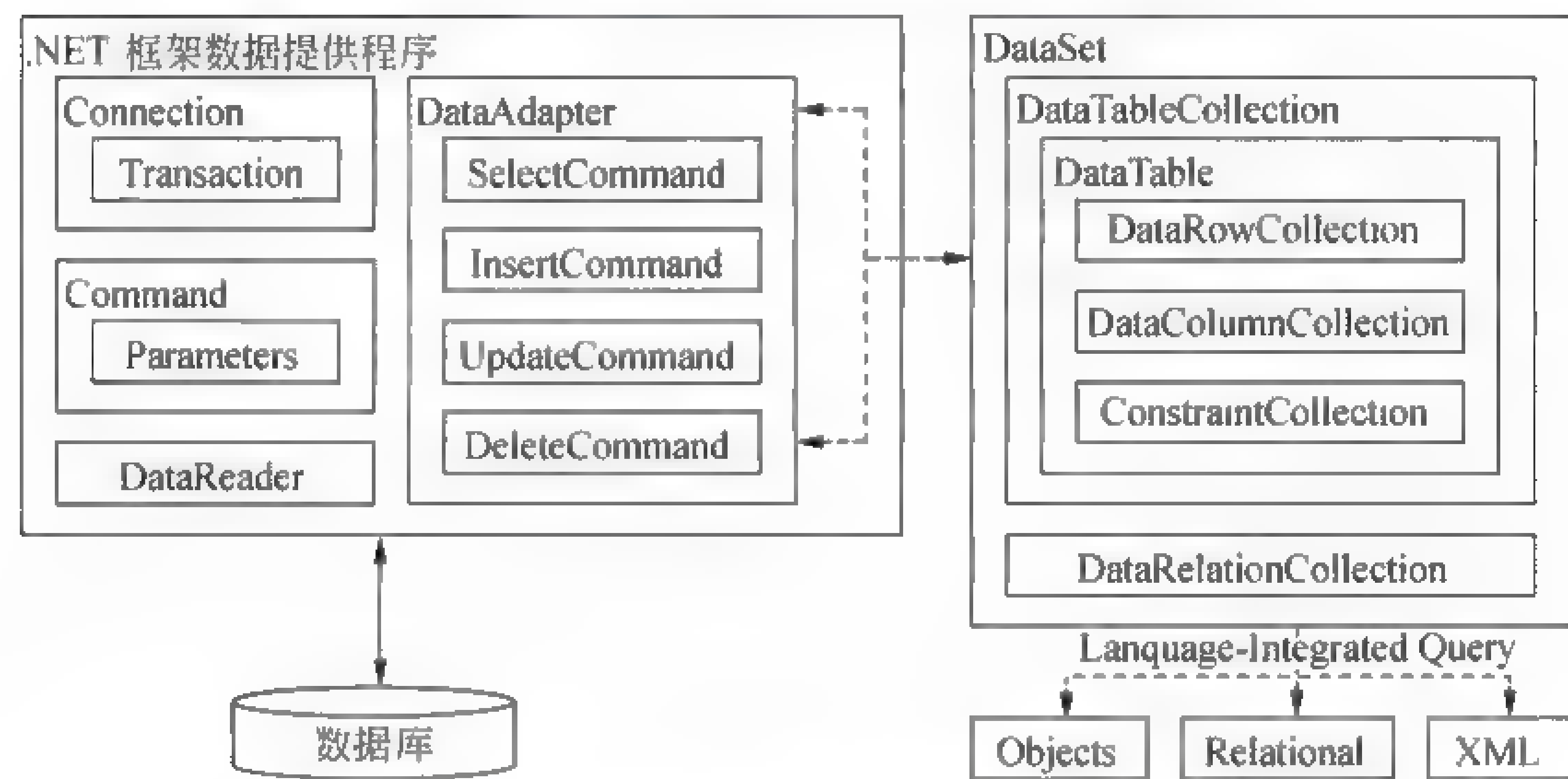

图 5-55　.NET 框架提供程序与 DataSet 间的关系

ADO.NET 类在 System.Data.dllt 中。使用 SQL Server 的方法是 using System.Data.SqlClient，并且它能与 System.Xml.dll 中的 XML 类集成。具有断开式数据结构，能够与 XML 紧密集成，具有能够组合来自多个不同数据源的数据的通用数据表示形式，以及具有为与数据库交互而优化的功能。

ADO.NET 实体框架使开发人员可以编写更少的数据访问代码，减少维护，将数据结构抽象化为更易于开展业务的方式，并且有利于数据的持久性。通过 EntityClient 可与概念层的实体数据模型（EDM）交互，如图 5-56 所示。

5.8.4　pureXML

pureXML 是一个新的 DB2 9 特性，它能够在数据库表中以原有格式存储 XML 数据，如图 5-57 所示。当然如 Oracle 也提供了 Oracle XML DB 用于处理 XML，SQLsever 2000 以后的版本也提供了对 XML 的支持。

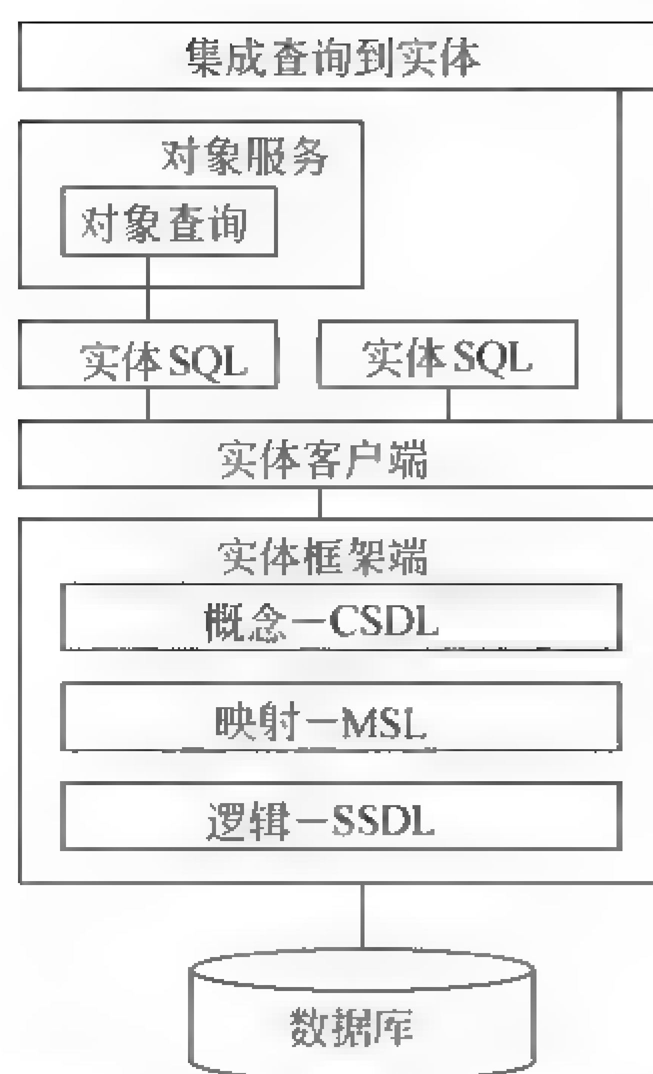

图 5-56　ADO.NET 实体框架

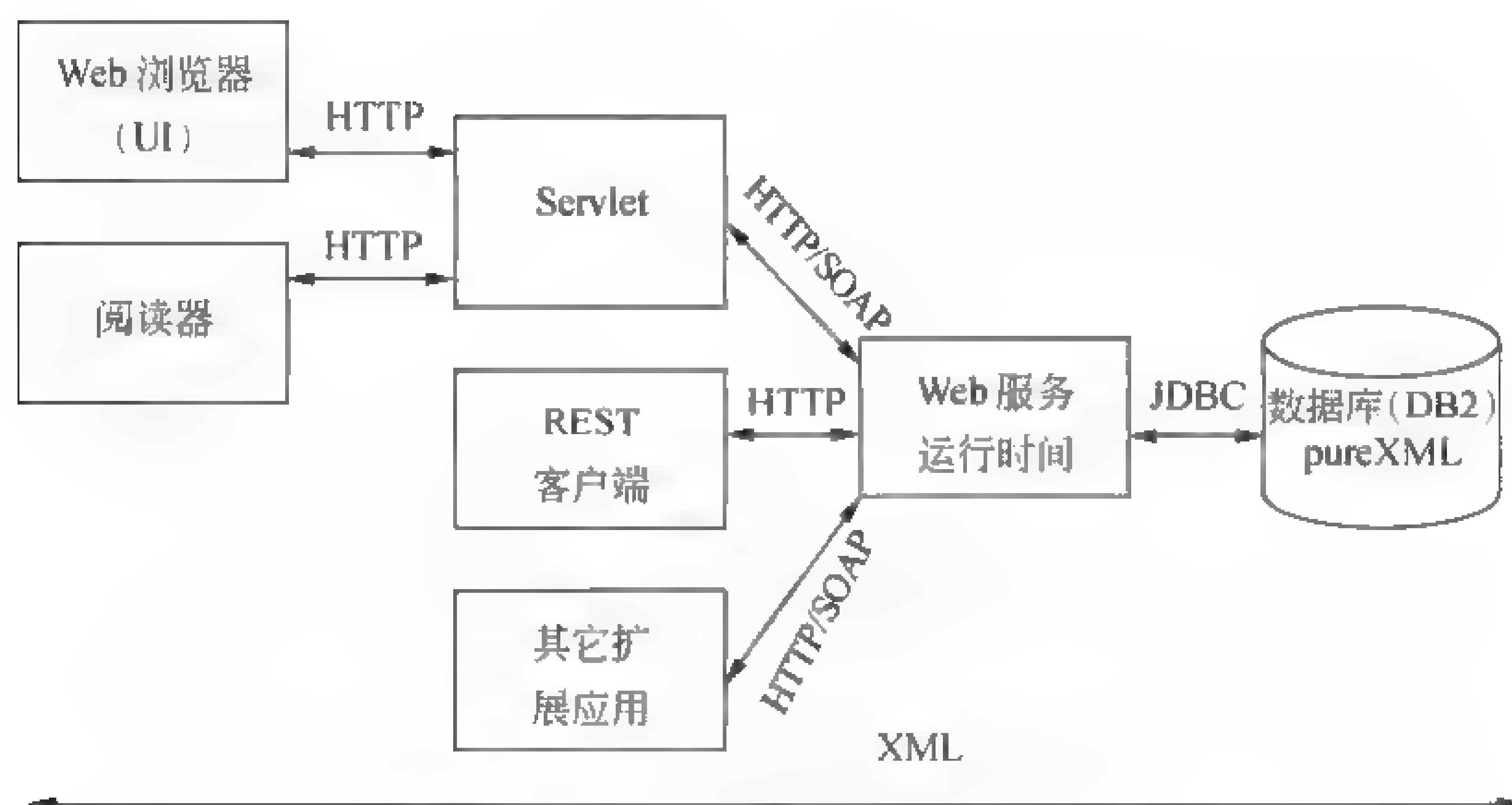

图 5-57　端到端的 XML

下面以 DB2 的 pureXML 为例进行简介。

DB2 pureXML 可以存储、更新、删除、查询和索引格式良好的 XML。通过将 XPath、XQuery 组合进行查询，使得用户可以检索整个 XML 文档或文档片断。DB2 9 中的 pureXML 技术包含以下功能：

（1）pureXML 数据类型和存储技术，能够高效地管理 XML 文档中常见的层次化结构。

（2）pureXML 索引技术，能够快速地搜索 XML 文档的子树。

（3）基于行业标准的新的查询语言支持（XQuery 和 SQL/XML）和新的查询优化技术。

（4）对管理、检验和演化 XML 模式的支持。

（5）全面的管理功能，包括对流行的数据库实用程序的扩展。

（6）与流行的应用程序编程接口（API）和开发环境的集成。

（7）XML 分解和发布功能，能够处理现有的关系模式。

（8）保持了 DB2 原有的可靠性、可用性、可伸缩性、性能、安全性和成熟度。

同时，作为 pureXML 的固有部分，DB2 9 为开发以 XML 为中心的和混合型的应用程序提供了丰富的支持，能够简化编码、减少开发时间和改进应用程序修改的敏捷性。DB2 9 中的 XML 开发支持包括：

（1）支持常用的编程语言和应用程序接口：语言：C/C++、Java™、C#、Visual Basic®、COBOL、PHP。接口：JDBC™、CLI / ODBC、.NET、Embedded SQL、SQLJ。

（2）支持使用 XQuery 和 SQL/XML（或同时使用这两种技术）查询数据。

（3）提供全面的 XML 功能（包括 XQuery 构建器）。

（4）与 Visual Studio®和.NET 紧密地集成。

（5）提供新的代码示例，改进了 DB2 SAMPLE 数据库，如下代码片段就是使用 DB2 提供的 JDBC 扩展访问 XML 数据：

```
com.h2.jcc.DB2Xml xml1 =
(com.h2.jcc.DB2Xml) rs.getObject ("xml stuff");
String s = xml1.getDB2String();
InputStream is = xml1.getDB2XMLBinaryStream("UTF-16");
```

（6）支持 Import/Export：把 XML 文档导入和导出数据库，Backup/Restore：与关系数据一起备份和恢复 XML 数据，Runstats：更新查询优化所用的统计数据，High Availability and Disaster Recovery (HADR)：使用辅助或离站系统，高级安全性：包括细粒度的基于标签的访问控制，存储过程：能够通过参数传递 XML 文档。

pureXML 可以使同一个数据服务器同时存储关系和 XML 数据，这种数据库技术先进、容易使用、基于标准而且，已经经过实践检验。表 5-11 所示是对这几种数据库访问技术的比较。

表 5-11 几种常用的数据库访问技术比较

数据库访问技术	技术来源	处理数据结构	表示形式	访问数据库	支持语言
ODBC	微软	结构化	API	多种数据库	SQL
JDBC	Sun	结构化	API	多种数据库	SQL
ADO.NET	微软	结构/非结构化	Recordset	多种数据库	SQL\XML

小　结

在这一章概述、总结分析了面向开源软件的开发技术，这些技术包括常用的开发语言、所支持的开发环境、所满足的服务器软件、Web 2.0 技术、面向服务的软件开发技术、语义化的软件技术和数据库访问技术。首先对这些技术进行全面总结，使其成为一种体系化的知识结构，并采用一些易于理解的个体程序案例进行分析。这样使本章内容更为丰富、更易于掌握。下面一章将在这些相关的软件开发技术之上，进一步概述、总结分析面向开源软件的软件开发框架。

第 6 章　面向开源软件的软件开发开源框架

开源软件是通过聚集群体智慧来促进软件的框架更为合理、程序代码更为优美，以及来提高软件可重用性。但开源软件的研发是离不开基本的软件研发结构、方法和语言等，因此，在前面的章节进行全面的分析了面向开源软件的软件开发基础设施，这一章继续分析当前流行的开源软件框架，为向开源软件的软件开发方法、技术提供策略。

6.1　概述

开源软件技术通常表现形式是以轻量级框架的模式进行应用的，所谓轻量级框架是相对于重量级框架而言的一种程序设计模式；与重量级框架相比，解决问题的侧重点是不同的。目前Struts、Spring、Hibernate组成的框架是最常用的轻量级框架之一，该框架侧重于减小开发的复杂度，但它处理诸如如事务功能弱、不具备分布式处理能力使有所减弱，因此比较适用于开发中小型企业应用，但随着开源软件的发展和开源软件框架各类增多，开源软件也逐渐成为大中型企业的企业系统的选择，或将开源软件与商业软件联合使用来做为企业级的应用选择。

采用轻量框架一方面因为尽可能的采用基于POJOs的方法进行开发，使应用不依赖于任何容器，这可以提高开发调试效率；另一方面轻量级框架多数是开源项目，开源社区提供了良好的设计和许多快速构建工具，以及大量现成可供参考的开源代码，这有利于项目的快速开发。例如目前Tomcat + Struts + Spring + Hibernate + MySQL，Lucene，Kettle，Axis，CXF，LAMP[Linux + Apache + MySQL +（Perl、PHP、Python）]等已经成为许多开发者开发J2EE中小型企业应用偏爱的一种架构选择。

随着可供选择的框架层出不穷，开发者可以根据需要对应于企业应用三个层次的轻量级框架实行选择，如表示层可以选择Struts、业务逻辑层可以选择Spring、数据持久层可以选择Hibernate等。而作为重量级框架EJB框架则强调高可伸缩性、容器封装性，适合与开发大型企业应用。在EJB体系结构中，一切与基础结构服务相关的问题和底层分配问题都由应用程序容器或服务器来处理，且EJB容器通过减少数据库访问次数和分布式处理等方式提供了专门的系统性能解决方案，能够充分解决系统性能问题，但EJB实现复杂、容器间的通信对初学者来说是难以把握的。但轻量级框架的产生并非是对重量级框架的否定，甚至在某种程度上可以说二者是互补的。轻量级框架在努力发展以开发具有更强大，功能更完备的企业应用；而新的EJB规范EJB 3.0则在努力简化J2EE的使用以使得EJB不仅仅是擅长处理大型企业系统，也利用开发中小型系统，这也是EJB轻量化的一种努力。对于大型企业应用，以及将来可能涉及到能力扩展的中小型应用来说，采用结合使用轻量级框架和重量级框架也不失为一种较好的解决方案。

面向轻量级的软件开发通常与一套开发方法、框架和设计原理一起使用，即不同的功能需求采用不同的开源软件，主要采用以下策略来实行轻量级选择：

（1）合并过程、选择技术和原理的策略，优先选择较简单的技术；

（2）在一个稳固、轻量级的基础上进行构建，并尽量争取最可能的透明性；

（3）可以利用的技术，如依赖注入和 AOP 形式来规划开源软件。

主要表现如下：

（1）轻量级方法包括敏捷过程，例如极限编程（XP）和 Scrum。它们强调开发中测试第一，积极调动客户和重构。

（2）轻量级框架鼓励人们使用简单原始的 Java 对象（POJO）编程，而不是类似 EJB 的重量级、面向组件的模型。

（3）轻量级设计模式可以在对象和集成服务之间进行松散耦合，而无须艰苦地编写业务逻辑或领域模型。

同时，轻量级与重量级软件开发框架也具有一个共同的特点：

（1）基于 POJO 的编程——轻量级容器不具侵犯性。它不强迫执行任何 API。

（2）生命周期管理——轻量级容器管理放入其中对象的生命周期。在最低限度下，它们实例化并销毁对象。

（3）依赖性解析——轻量级容器提供了一个普通的依赖性解析策略。并且现在多数容器支持依赖注入的策略。还有一些支持 J2EE 风格的策略，称为服务定位。

（4）一致的配置——轻量级容器是一个便于提供一致配置服务的位置。

（5）服务关联——轻量级容器提供一种将服务与容器中的对象相关联的方法。

轻量级容器有许多胜于其他容器架构的优点。例如，可以使用一个更加简单、基于 POJO 的编程模型。使用 POJO 编程，应用程序会更加易于测试。而且对象也可以在容器外运行。例如，在一个测试用例中。通过依赖注入，轻量级容器减少了组件间的依赖性。

6.2 DWR

DWR（Direct Web Remoting）是一个用于改善 web 页面与 Java 类交互的远程服务器端 Ajax 开源框架，可以帮助开发人员开发包含 AJAX 技术的网站。它还可以允许在浏览器里的代码使用运行在 WEB 服务器上的 JAVA 函数，就像它就在浏览器里运行一样。

6.2.1 AJAX 基本应用方法

前面章节已经概述了 AJAX 的基本原则，这一节主要简介 AJAX 的基本应用方法，它是一种广泛应用在浏览器的网页开发技术，即 AJAX 一般是由 Ajax 由 HTML、JavaScript 技术、DHTML 和 DOM 组成。AJAX 的应用是将 Web 浏览器作为运行平台。这些浏览器目前包括：Internet Explorer、Mozilla、Firefox、Opera、Konqueror 及 Mac OS 的 Safari。但是 Opera 不支持 XSL 格式对象，也不支持 XSLT。

6.2.1.1 AJAX 的基本操作

AJAX 是由 Jesse James Garrett 发明的[1]，即运用 XHTML + CSS 来表达信息，运用

① http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications

JavaScript 操作 DOM（Document Object Model）运行动态效果，运行 XML 和 XSLT 进行数据交换及操作，运用 XMLHttpRequest 为 Agent 与网页服务器进行异步数据交换，运用 JavaScript 技术实现数据访问。图 6-1 所示是 AJAX 运行结构。

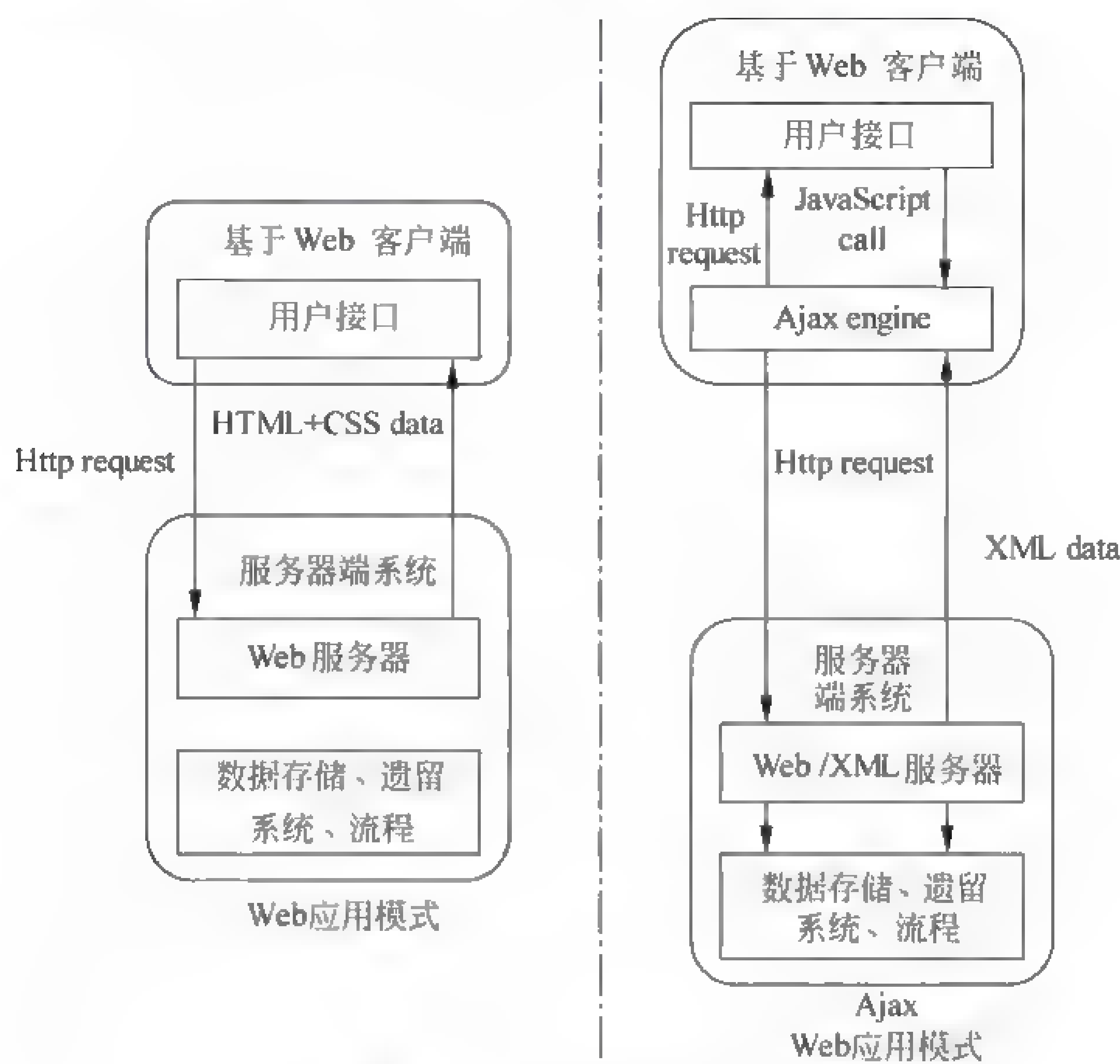

图 6-1　AJAX 运行结构

1. XMLHttpRequest

XMLHttpRequest 是一个 JavaScript（JavaScript 不要求指定变量类型）对象，它提供了下面几个方法和属性：

（1）open()：建立到服务器的新请求。具有五个参数：request-type（发送请求的类型。类型的值是 GET 或 POST，但也可以发送 HEAD 请求）、url（要连接的 URL）、asynch（如果希望使用异步连接则为 true，否则为 false。该参数是可选的，默认为 true）、username（如果需要身份验证，则可以在此指定用户名。该可选参数没有默认值）、password（如果需要身份验证，则可以在此指定口令。该可选参数没有默认值）。

（2）send()：向服务器发送请求。

（3）abort()：退出当前请求。

（4）readyState：提供当前 HTML 的就绪状态。

（5）responseText：服务器返回的请求响应文本。

Ajax 应用程序中的 HTTP 中需要五种就绪状态：0[请求没有发出（在调用 open()之前）]；1[请求已经建立但还没有发出（调用 send()之前）]；2[请求已经发出正在处理之中（这里通常可以从响应得到内容头部）]；3（请求已经处理，响应中通常有部分数据可用，但是服务器还没有完成响应）；4（响应已完成，可以访问服务器响应并使用它）。还需要检查 HTTP 状态，期望的状态码是 200，它表示一切顺利。如果就绪状态是 4 而且状态码是 200，就可以

处理服务器的数据了，而且这些数据应该就是要求的数据（而不是错误或者其他有问题的信息）。401（未经授权）、403（禁止）、404（没找到）、200（一切正常）、301（永久移动）、302[找到（即请求被重新定向到另外一个 URL/URI 上）]、305[使用代理（请求必须使用一个代理来访问所请求的资源）]，因此，还要在回调方法中增加状态检查。如下程序片段是 XMLHttpRequest 的基本操作方法：

```
<body>
<input type="text" size="14" name="phone" id="phone"
        onChange="getCustomerInfo();" />
</body>

<script language="javascript" type="text/javascript">
var request;
function createRequest() {
//建立 XMLHttpRequest 方法
  try {
    request = new XMLHttpRequest();
  } catch (trymicrosoft) {
//对 Microsoft 浏览器的支持
    try {
      request = new ActiveXObject("Msxml2.XMLHTTP");
    } catch (othermicrosoft) {
      try {
        request = new ActiveXObject("Microsoft.XMLHTTP");
      } catch (failed) {
        request = false;
      }
    }
  }
  if (!request)
    alert("Error initializing XMLHttpRequest!");
}
function getCustomerInfo() {
  createRequest();
  //创建方法

function getCustomerInfo() {
     var phone = document.getElementById("phone").value;
     var url = "/cgi-local/lookupCustomer.jsp?phone=" + escape(phone);
     //查询客户电话号码
     //建立请求 URL
     request.open("GET", url, true);
     //打开请求
     request.onreadystatechange = updatePage;
     //设备回调方法
```

```
        request.send(null);
        //发送请求
    function updatePage() {
        alert("Server is done!");
      }
    //设置回调方法
    function updatePage() {
        if (request.readyState == 4)//HTTP 就绪状态
          if (request.status == 200)//检查 HTTP 状态码
            alert("Server is done!");
            else if (request.status == 404)
            alert("Request URL does not exist");
            else if (request.status == 403)
            alert("Access denied.");
          else
            alert("Error: status code is " + request.status);
          //错误检查
            if(request.readyState != 4 and request.status != 200){
            var response = request.responseText.split("|");
            document.getElementById("order").value = response[0];
            document.getElementById("address").innerHTML =
              response[1].replace(/\n/g, "");
            } else
            alert("status is " + request.status);
            }
            }
            //处理服务器响应
      }
    }
    </script>
```

2. 利用 DOM 进行 Web 响应

DOM（Document Object Model）是 W3C 的一个规范，是 Ajax 应用程序的一个主要部分。它定义了对象的类型和属性，其对象表示是 DOM 树，DOM 树的结点是 DOM 中最基本的对象类型，且基本上一切都是结点，每个元素在最底层上都是 DOM 树中的结点，而且每个属性都是结点。DOM 结点的属性主要有：

（1）nodeName：报告结点的名称。

（2）nodeValue：提供结点的“值”。

（3）parentNode：返回结点的父结点。且每个元素、属性和文本都有一个父结点。

（4）childNodes：是结点的孩子结点列表。对于 HTML，该列表仅对元素有意义，文本结点和属性结点都没有孩子。

（5）firstChild：仅仅是 childNodes 列表中第一个结点的快捷方式。

（6）lastChild：是另一种快捷方式，表示 childNodes 列表中的最后一个结点。

（7）previousSibling：返回当前结点之前的结点。换句话说，它返回当前结点的父结点的

childNodes 列表中位于该结点前面的那个结点。

（8）nextSibling：类似于 previousSibling 属性，返回父结点的 childNodes 列表中的下一个结点。

（9）attributes：仅用于元素结点，返回元素的属性列表。

DOM 的结点方法主要有：

（1）insertBefore(newChild, referenceNode)：将 newChild 结点插入到 referenceNode 之前。且应该对 newChild 的目标父结点调用该方法。

（2）replaceChild(newChild, oldChild)：用 newChild 结点替换 oldChild 结点。

（3）removeChild(oldChild)：从运行该方法的结点中删除 oldChild 结点。

（4）appendChild(newChild)：将 newChild 添加到运行该函数的结点之中。newChild 被添加到目标结点孩子列表中的末端。

（5）hasChildNodes()：在调用该方法的结点有孩子时则返回 true，否则返回 false。

（6）hasAttributes()：在调用该方法的结点有属性时则返回 true，否则返回 false。

Web 应用程序中只用到四种结点类型：

（1）文档结点表示整个 HTML 文档。

（2）元素结点表示 HTML 元素，如 a 或 img。

（3）属性结点表示 HTML 元素的属性，如 href（a 元素）或 src（img 元素）。

（4）文本结点表示 HTML 文档中的文本。

在 DOM 中，虽然会大量使用元素结点，但很多需要对元素执行的操作都是所有结点共有的方法和属性，而不是元素特有的方法和属性。元素只有两组专有的方法：

（1）与属性处理有关的方法：

① getAttribute(name) 返回名为 name 的属性值。

② removeAttribute(name) 删除名为 name 的属性。

③ setAttribute(name, value) 创建一个名为 name 的属性并将其值设为 value。

④ getAttributeNode(name) 返回名为 name 的属性结点。

⑤ removeAttributeNode(node) 删除与指定结点匹配的属性结点。

（2）与查找嵌套元素有关的方法：getElementsByTagName(elementName)返回具有指定名称的元素结点列表。

（3）通常使用元素类的方法处理属性：

① getAttribute(name) 返回名为 name 的属性值。

② removeAttribute(name) 删除名为 name 的属性。

③ setAttribute(name, value) 创建一个名为 name 的属性并将其值设为 value。

（4）DOM 结点类型所定义的常用一些常量：

① Node.ELEMENT_NODE 是表示元素结点类型的常量。

② Node.ATTRIBUTE_NODE 是表示属性结点类型的常量。

③ Node.TEXT_NODE 是表示文本结点类型的常量。

④ Node.DOCUMENT_NODE 是表示文档结点类型的常量。

如下程序片段就是利用 DOM 进行 Web 响应的例子：

```
<html>
 <head>
  <title>利用 DOM 进行 Web 响应</title>
  <script language="JavaScript">
   function test() {
    var myDocument = document;
    var htmlElement = myDocument.documentElement;
    alert("The root element of the page is " + htmlElement.nodeName);
    var pElement = myDocument.createElement("p");
    var text = myDocument.createTextNode("Here's some text in a p element.");
    pElement.appendChild(text);
    bodyElement.appendChild(pElement);
    //文档结点
    var headElement = htmlElement.getElementsByTagName("head")[0];
    if (headElement != null) {
      alert("We found the head element, named " + headElement.nodeName);
      var titleElement = headElement.getElementsByTagName("title")[0];
      if (titleElement != null) {
         var titleText = titleElement.firstChild;
         alert("The page title is '" + titleText.nodeValue + "'");
      }
      var bodyElement = headElement.nextSibling;
      while (bodyElement.nodeName.toLowerCase() != "body") {
        bodyElement = bodyElement.nextSibling;
      }
     if (bodyElement.hasChildNodes()) {
        for (i=0; i<bodyElement.childNodes.length; i++) {
          var currentNode = bodyElement.childNodes[i];
          if (currentNode.nodeName.toLowerCase() == "img") {
            bodyElement.removeChild(currentNode);
          }
        }
      }
    }
  }
  </script>
 </head>
 <body>
  <p>JavaScript and DOM are a perfect match.
     You can read more in <i>Head Rush Ajax</i>.</p>
  <img src="http://localhost/Images/hraj cover-150.jpg" />
  <input type="button" value="Test me!" onClick="test();" />
 </body>
</html>
```

3．请求和响应中的 XML

XML 被认为是 Ajax 底层的核心技术之一，而且 XMLHttpRequest 也支持这种用法，但请求是 HTTP 而非 XML；XML 在 AJAX 中只做为一种请求与响应的应用模式。因此，XML 在异步应用程序中有以下两种基本的用法。

（1）以 XML 格式从网页向服务器发送请求。即用 XML 发送请求，需要将请求的格式设置为 XML，这时可以使用 API 来完成，也可以与文本连成字符串，然后将结果发送到服务器。

（2）以 XML 格式在网页中从服务器接收请求。用 XML 接收请求，需要从服务器上接收响应，然后从 XML 提取数据（同样，可以用 API）。这种情况下，关键在于来自服务器的数据，而恰好需要从 XML 中提取这些数据以便使用。作为一个 XML 文档，由一个 DOM Document 对象表示。

如下程序片段就是请求与响应 XML 的代码：

```
<script language="JavaScript">
function callServer() {
  var firstName = document.getElementById("firstName").value;
  var lastName = document.getElementById("lastName").value;
  var street = document.getElementById("street").value;
  var city = document.getElementById("city").value;
  var state = document.getElementById("state").value;
  var zipCode = document.getElementById("zipCode").value;
  var xmlString = "<profile>" +
    "  <firstName>" + escape(firstName) + "</firstName>" +
    "  <lastName>" + escape(lastName) + "</lastName>" +
    "  <street>" + escape(street) + "</street>" +
    "  <city>" + escape(city) + "</city>" +
    "  <state>" + escape(state) + "</state>" +
    "  <zip-code>" + escape(zipCode) + "</zip-code>" +
    "</profile>";
  var url = "/scripts/saveAddress.jsp";
  xmlHttp.open("POST", url, true);
  xmlHttp.setRequestHeader("Content-Type", "text/xml");
  xmlHttp.onreadystatechange = confirmUpdate;
  xmlHttp.send(xmlString);
}
//用 XML 发送名/值对
function updatePage() {
  if (request.readyState == 4) {
    if (request.status == 200) {
      var xmlDoc = request.responseXML;
      var showElements = xmlDoc.getElementsByTagName("show");
      for (var x=0; x<showElements.length; x++) {
        var title = showElements[x].childNodes[0].value;
        var rating = showElements[x].childNodes[1].value;
        //遍历所有 show 元素
```

```
        }
      }
    }
  }
  //获取所有 show 元素
  </script>
```

4．JSON 数据传输

JSON（JavaScript Object Notation）可以将 JavaScript 对象中表示的一组数据转换为字符串，然后就可以在函数之间轻松地传递这个字符串，或者在异步应用程序中将字符串从 Web 客户机传递给服务器端程序。这个字符串看起来有点儿古怪（但是 JavaScript 很容易解释它，而且 JSON 可以表示比名称/值对更复杂的结构（如{ "firstName": "Brett" }）。同时 JSON 是 JavaScript 原生格式，这意味着在 JavaScript 中处理 JSON 数据不需要任何特殊的 API 或工具包。如下程序片段就是 JSON 定义和使用方法：

```
<script language="JavaScript">
var people =
  { "programmers": [
    {     "firstName":     "Brett",     "lastName":"McLaughlin",     "email":
    "brett@newInstance.com" },
    {      "firstName":      "Jason",      "lastName":"Hunter",      "email":
    "jason@servlets.com" },
    {     "firstName":     "Elliotte",     "lastName":"Harold",     "email":
    "elharo@macfaq.com" }
   ],
  "authors": [
    { "firstName": "Isaac", "lastName": "Asimov", "genre": "science fiction" },
    { "firstName": "Tad", "lastName": "Williams", "genre": "fantasy" },
    {  "firstName":  "Frank",  "lastName":  "Peretti",  "genre":  "christian
    fiction" }
   ],
  "musicians": [
    { "firstName": "Eric", "lastName": "Clapton", "instrument": "guitar" },
    {  "firstName":  "Sergei",  "lastName":  "Rachmaninoff",  "instrument":
    "piano" }
   ]
  }
 people.programmers[0].lastName; //访问 people
 String newJSONtext = people.toJSONString(); //转换回字符串
 String myObjectInJSON = myObject.toJSONString();
//GET 发送给服务器
 var url = "organizePeople.jsp?people=" + escape(people.toJSONString());
 request.open("GET", url, true);
 request.onreadystatechange = updatePage;
 request.send(null);
```

```
//利用 POST 发送给服务器
 var url = "organizePeople.jsp?timeStamp " + new Date().getTime();
 request.open("POST", url, true);
 request.onreadystatechange = updatePage;
 request.setRequestHeader("Content-Type", "application/x-www-form-
 urlencoded");
 request.send(people.toJSONString());
</script>
```

6.2.1.2 AJAX 的持久对象映射

Persevere JavaScript[①]是一种面向支持正交持久性的 JavaScript 语言的远程持久对象映射框架，它将 JavaScript 对象映射到远程持久化的数据存储区，其方法是使用 JSON 和符合 JSON for persistence（JSPON）规范的标准，且用状态传输（Representational State Transfer，REST）HTTP 来实现。从而使 Persevere 持久对象框架为浏览器 JavaScript 环境带来了持久对象映射功能，还能自动化基于 Asynchronous JavaScript + XML（Ajax）的 Web 应用程序中的映射和通信。并且动态使 JavaScript 语言在本质上就很适合将对象映射到持久数据。

Persevere JavaScript 提供了客户机和 Persevere 服务器，该服务器自动化了将数据库和其他持久存储作为 JSON REST 服务，并加以公开的过程。并借助 Persevere，通过在浏览器中访问和修改 JavaScript 对象就可以访问和修改服务器上的数据。Persevere 也会对必要的序列化、Ajax 调用和去序列化进行自动化处理。同时，可以使用常规的 JavaScript 对象和属性来访问和操纵持久化的数据。持久数据源是 Persevere 服务器提供的默认数据库存储区。Persevere 服务器还提供了在服务器上将数据库映射为 JSON 格式的功能，Persevere 客户机和服务器通过这种格式相互通信。图 6-2 所示是持久 SOA 的体系结构。

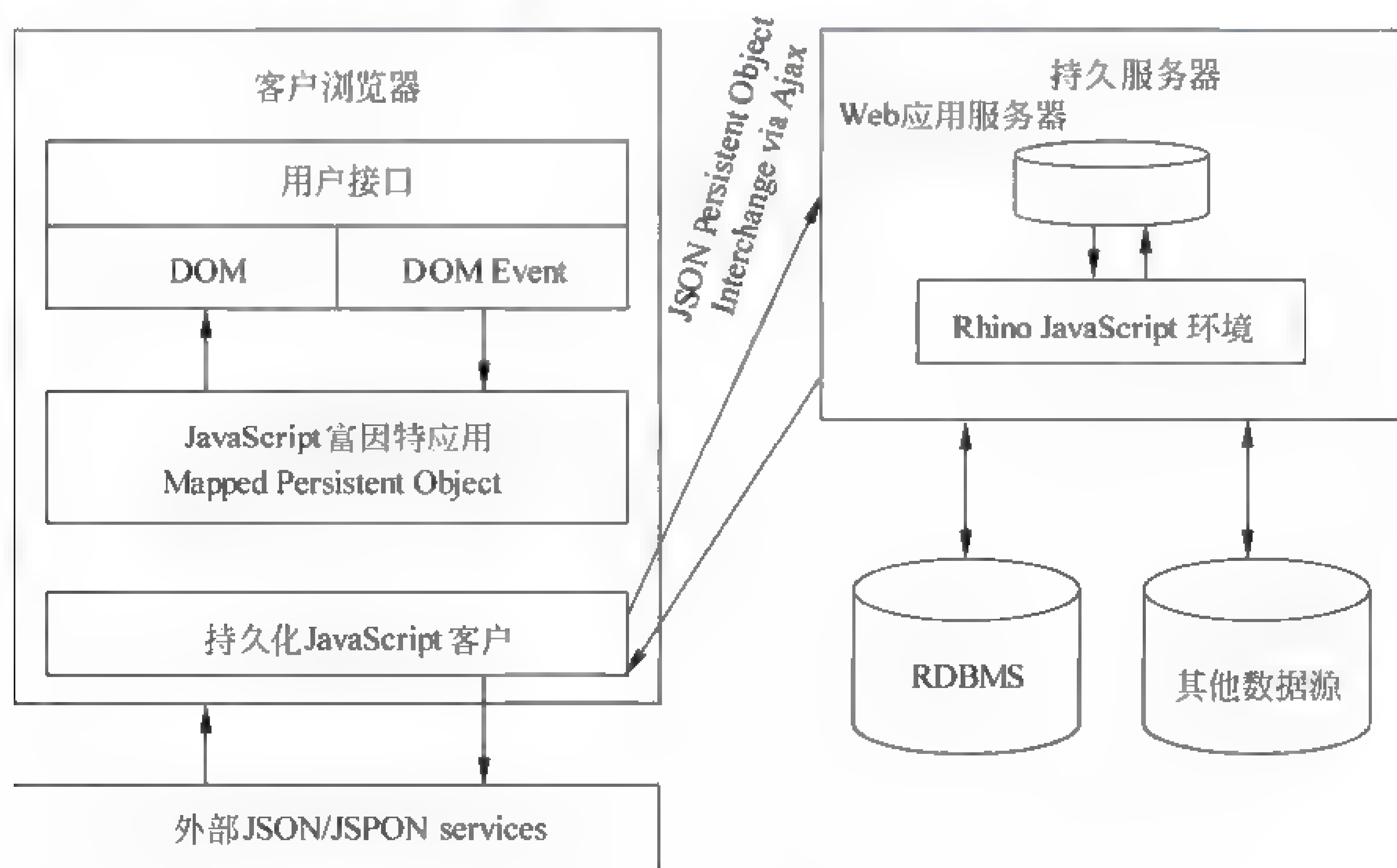

图 6-2　Persevere 持久 JSON SOA

① http://code.google.com/p/persevere-framework

有几种方式可用来访问 Persevere 的持久功能。但通过标准 JavaScript 属性语法支持属性的延迟加载时，以及需要自动保存属性更改时，都需要进行预处理。然而，即使没有使用持久 JavaScript API 进行过预处理，具有显式保存和延迟加载功能的完全持久对象映射仍可用。而且，可以在服务器上或构建过程中执行预处理。为了获得最佳性能，应该对最终的应用程序进行预处理。或者，也可以在 JavaScript V1.7 中本地运行 Persevere 的全部特性（除了预处理），而对不具有 JavaScript V1.7 支持的浏览器则也可以使用预处理。

可以使用 Persevere 框架访问持久对象。在使用 Persevere 框架时，必须预处理 JavaScript 代码以添加延迟加载和正交持久性支持。最简便的实现方法是调用 persevere.loadScript 来加载 JavaScript 文件。对于 Persevere，可以使用.pjs（而非.js）扩展名来表明该文件是一个具备全部持久性支持的 JavaScript 文件。loadScript()函数然后执行必要的处理。

Persevere 服务器支持传统的关系数据库，也支持通过 JSON REST 服务公开 SQL 数据库表数据，以使用 Persevere 进行持久对象映射。若要查看 SQL 数据库表的例子，可以打开持久对象浏览器并加载 exampledatabasetable/对象。也可以利用 load()函数从 JavaScript 文件载入该数据，并且此表可以作为一个常规的 JavaScript 数组被访问。如下程序片段是 JavaScript 与关系数据库的交互：

```
function changeNameForFirstRow() {
  pjs.load("exampledatabasetable/")[0].name="New name";
 }
```

该语句从表中加载数据并将第一行中的名称栏的值改为一个新值。Persevere 也提供了对事务进行控制的功能。Persistent JavaScript API 为控制事务定义了两种方法：pjs.commit()和 pjs.rollback()。commit()方法会立即尝试提交已发生的改变。rollback()方法则取消从事件开始以来或最后一次提交的所有更改，程序片段如下：

```
function addBlogPost() {
  ...
  if (!confirm("Are you sure you want to create this post?"))
    pjs.rollback();
 }
```

6.2.1.3　Ajax 和 Web 服务

Web 服务是一种以独立于语言（及平台）的方式公开功能的好方法。AJAX 是一种访问其他资源内容而无须在当前 Web 页调用新请求的技术方法。Web 开发人员结合 AJAX 与 Web 服务可以开发出功能强大的应用程序，这些程序利用最先进的技术，并提供增强的用户体验。在具体结合时，请求可以是简单的 HTTP 请求，且发送给公开的 Web 服务的也可以是 SOAP 消息。然后 Ajax 例程中的 JavaScript 端解析响应（也是 SOAP 格式，例如下程序片段 6-1）并提取所需数据返回给应用程序，然后展示给用户。

程序片段 6-1　SOAP 响应：

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

```
<SOAP ENV:Envelope
SOAP ENV:encodingStyle "http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<SOAP-ENV:Fault>
<faultcode xsi:type="xsd:string">SOAP-ENV:Client</faultcode>
<faultactor xsi:type="xsd:string" />
<faultstring xsi:type="xsd:string">method '' not defined in service</faultstring>
<detail xsi:type="xsd:string" />
</SOAP-ENV:Fault>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

下面的程序片段 6-2 中的函数名 invokeService 表示接收一个参数：type。这对应于 Web 服务操作（retrieveByType）接收的 type 参数。即 type 参数是包含 casting、trolling 或 other 的字符串。函数的前几行组装表示 SOAP 消息（相关 SOAP 的详细内容见第 5 章）。同时，需要注意 XML 其中一个元素直接对应操作名（retrieveByType），该元素的子元素根据页面文件中指定的参数名（type）来命名。并且那个元素值是传入该 JavaScript 函数的字符串参数，又称 type。接下来几行创建跨浏览器兼容请求对象，这是用来访问 Web 服务的对象。当请求对象创建后，函数设置回调函数，如本程序片段中是 populateDiv()函数，这是在网页上显示库存列表的函数。函数然后设置头部。创建与 SOAP 兼容的内容类型：text/xml。还需要注意 url 变量的使用。当创建的 Web 页面后，需要将变量指向自己 Web 服务的 URL。最后，使用请求对象发送 SOAP 消息。并且在服务不响应时可以设置超时。

程序片段 6-2　invokeService() JavaScript 函数：

```
<script language="JavaScript">
function invokeService(type) {//表示指定的参数名（type）来命名
 soapMessage = '<?xml version="1.0" encoding="ISO-8859-1"?>';
 soapMessage+='<SOAP-ENV:Envelope SOAP-ENV:encodingStyle=""';
 soapMessage+=' xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"';
 soapMessage+=' xmlns:xsd="http://www.w3.org/2001/XMLSchema"';
 soapMessage+=' xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">';
 soapMessage+=' <SOAP-ENV:Body> <ns1:retrieveByType xmlns:ns1="http:
 //fishinhole.com">';
 soapMessage+=' <type xsi:type="xsd:string">' + type + '</type>';
 soapMessage+=' </ns1:retrieveByType> </SOAP-ENV:Body> </SOAP-ENV:Envelope>';

 if(window.XMLHttpRequest) {
  httpRequest=new XMLHttpRequest();
 } else if (window.ActiveXObject) {
  httpRequest new ActiveXObject("Microsoft.XMLHTTP");
```

```
    }
    httpRequest.open("POST",url,true);
    if (httpRequest.overrideMimeType) {
     httpRequest.overrideMimeType("text/xml");
    }
    httpRequest.onreadystatechange=populateDiv;
     httpRequest.setRequestHeader("Man", url + " HTTP/1.1")
    httpRequest.setRequestHeader("MessageType", "CALL");
    httpRequest.setRequestHeader("Content-Type", "text/xml");
    httpRequest.send(soapMessage);
    valTimeout=setTimeout("timeout(httpRequest);",120000);
   }
   </script>
```

下面的程序片段 6-3 是 populateDiv() JavaScript 函数，在程序中的第 14 行以后的程序表示下一步创建一些 HTML，并以空 HTML 字符串开始。然后，将 SOAP 消息解析成 item 元素。Web 服务返回一个数组；这时，有可能 item 元素不止一个。可以用 for 循环来完成多个 item 元素。而对于每一个 item 元素，JavaScript 代码获取 item 元素的第一个子元素。在这个例子中，只有一个子元素，是个简单字符串。然后它提取该子元素的值，即库存中的一项。当 for 循环完成后，就有了完整列表。

程序片段 6-3　populateDiv() JavaScript 函数：

```
1 function populateDiv(){
2  try {
3     if(httpRequest.readyState==4) {
4       if(httpRequest.status==200) {
5          clearTimeout(valTimeout);
6          var text = httpRequest.responseText;
7      if (window.DOMParser) {
8       parser=new DOMParser();
9       xmlDoc=parser.parseFromString(text,"text/xml");
10      } else {
11         xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
12      xmlDoc.async="false";
13      xmlDoc.loadXML(text); }
14     var html = "";
15     for (i=0;i<xmlDoc.getElementsByTagName("item").length;i++) {
16      html += "<br/>" +
17           xmlDoc.getElementsByTagName("item")[i].childNodes[0].
              nodeValue;
18     }
19     var resultDiv=document.getElementById("resultDiv");
20     resultDiv.innerHTML = html;
21       }
22     }
```

```
23 } catch(e) {
24      alert("Error!"+e.description);
    }
  }
```

6.2.2 DWR 应用方法

AJAX 只能处理 JS（JavaScript）代码，而对于 JAVA 应用程序往往是 JavaBean，但 AJAX 不能直接调用 JavaBean。这时，DWR（Direct Web Remoting）开源软件就能有效解决这个问题，它可以屏蔽大多数不能直接访问的差别，从而提供一种有效的、易于编码的 Ajax 解决方案。DWR 是 Joe Walker 和 Mark Goodwin 的研究成果。它是 JavaScript 代码库与 Java servlet 的组合，它允许使用 Ajax 对服务器端 JavaBean 的方法进行透明的远程调用。DWR 是遵循 Apache Software License 2.0 发布的，图 6-3 是 DWR 如何简化 Ajax 编程结构图。

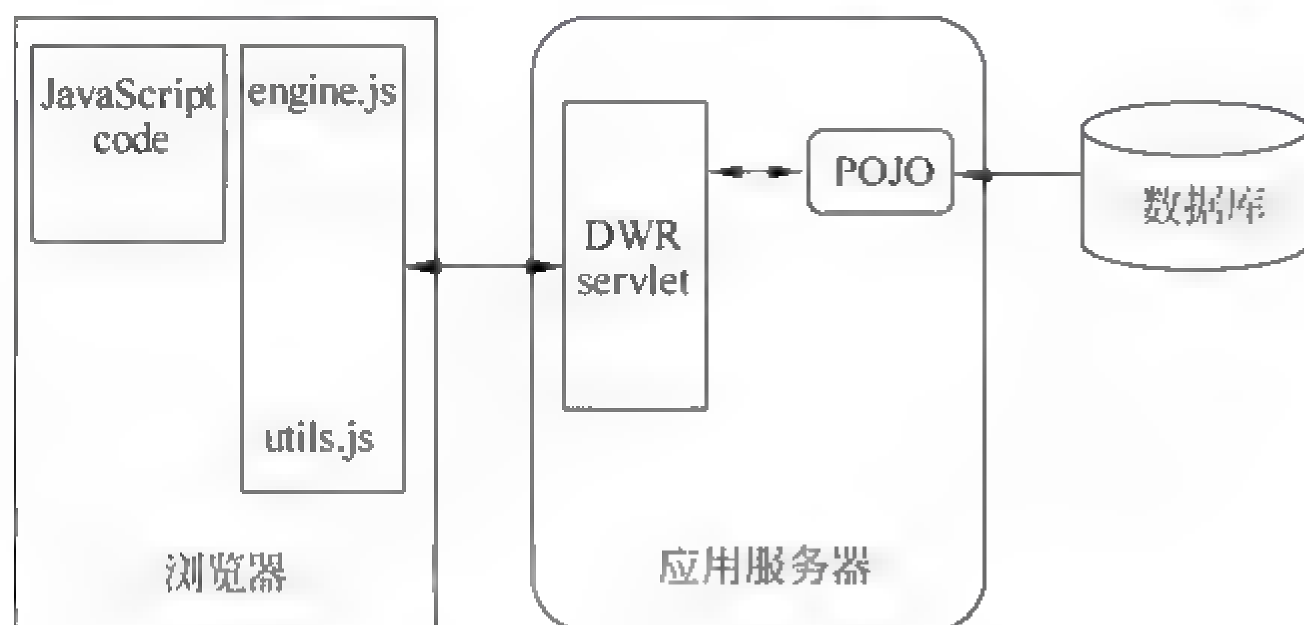

图 6-3　DWR 结构示意图

DWR 由以下两个组件组成：一个 JavaScript 库，它在用户的浏览器上运行（utils.js 和 engine.js）；一个 servlet，它运行在 Application Server 上。DWR 允许远程控制某些 JavaBean，使其 DWR 在该对象看上去好像可用于在用户浏览器上运行的 JavaScript 代码。DWR 还通过浏览器提供的 XMLHttpRequest 对象对在 Application Server 上运行的 servlet 进行异步调用。该 servlet 访问服务器端 JavaBean（一种 Plain Old Java Object（POJO））对象并通过采用 JSON 格式的 HTTP 响应发回所需的数据。JSON 格式是 JavaScript 友好的序列化格式，它允许 Web 浏览器中的 JavaScript 解释程序直接创建本机 JavaScript 对象，图 6-4 所示是一个基于 DWR 的具体应用结构。

在图 6-4 中，显示业务信息的页面现在是一个 HTML 页面，而不是一个 JSP 页面。Application Server 只作为 Web 服务器，向用户浏览器发送静态的网页。它不作为 JSP 容器。当 HTML 页面中的 JavaScript 代码访问 Java 对象，可以使用 DWR 对这些对象进行远程控制。服务器端 Java 对象不需要在传统的 Java EE 容器内被托管。在 Application Server 中有三个目的：充当静态的 Web 服务器并且服务于包含 Ajax 代码的静态 HTML 页面；支持 DWR 的执行；支持 servlet 以便显示信息。

DWR 使用 Java servlet 接受和响应 Ajax 请求，这个 servlet 包括在 DWR 库内，因此只需配置 Web 应用程序，以便在应用程序的 web.xml 文件内启用 servlet，如下程序片段就是在 web.xml 中配置 DWR：

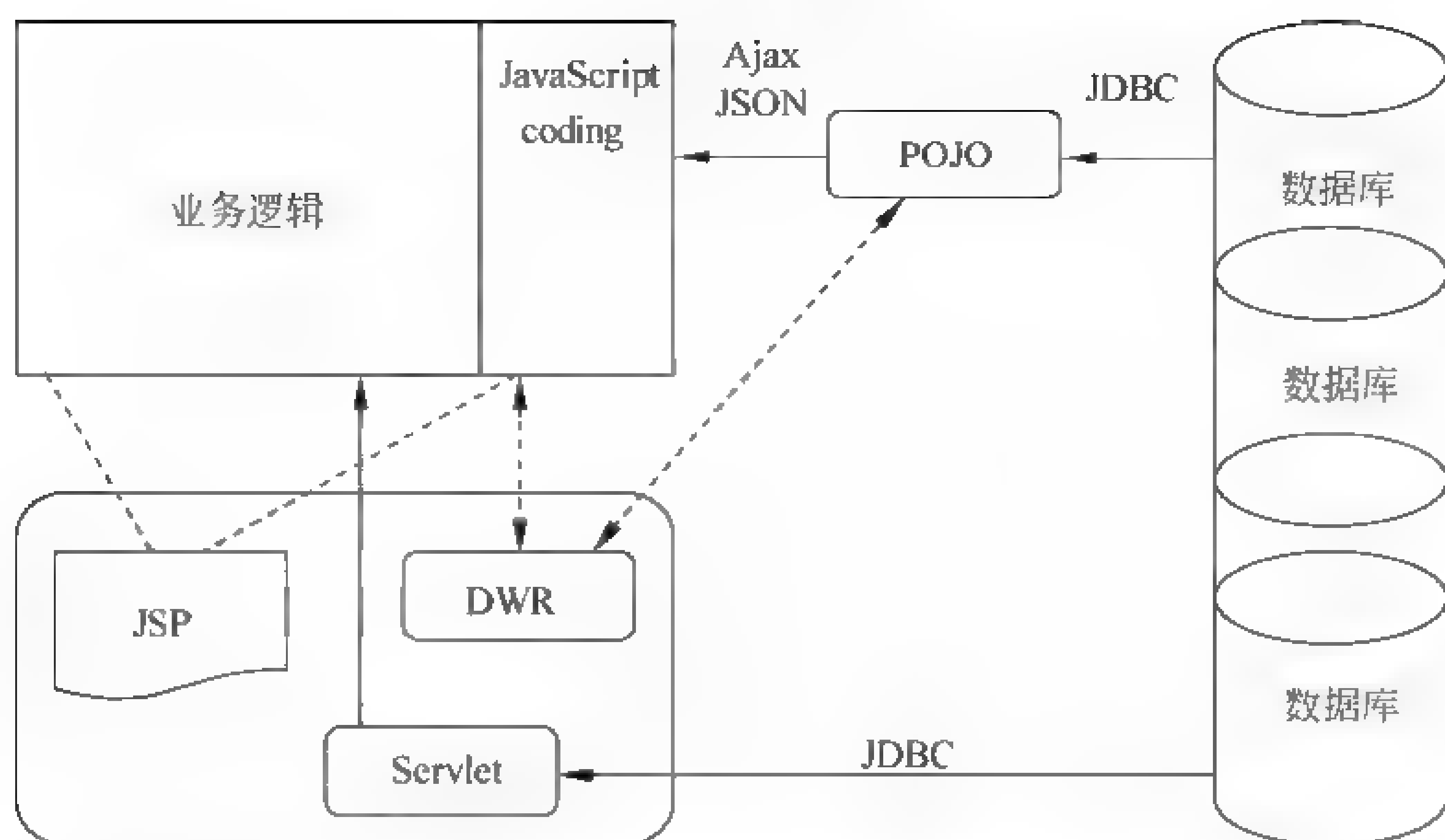

图 6-4　基于 DWR 的具体应用结构

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:web="http://java.sun.com/xml/ns/javaee/web-app 2 5.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-app 2 5.xsd"
    id="msgb" version="2.5">
    <display-name>在 web.xml 中配置 DWR</display-name>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
    </welcome-file-list>
    <servlet>
        <display-name>DWR Servlet</display-name>
        <servlet-name>dwr-invoker</servlet-name>
        <servlet-class>
            org.directwebremoting.servlet.DwrServlet
        </servlet-class>
        <init-param>
            <param-name>debug</param-name>
            <param-value>true</param-value>
        </init-param>
        <init-param>
            <param-name>classes</param-name>
            <param-value>org.test.dwrajax,… </param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>dwr-invoker</servlet-name>
        <url pattern>/dwr/*</url pattern>
    </servlet-mapping>
</web-app>
```

下面的程序片段表示在 Geronimo 中公布 DWR 2.0 库，即在 geronimo-web.xml 文件中，添加对 Application Server 中已经包含的默认的 DWR 2.0 版本的库的依赖项。

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.2"
        xmlns:nam="http://geronimo.apache.org/xml/ns/naming-1.2"
        xmlns:sec="http://geronimo.apache.org/xml/ns/security-1.1"
        xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.2">
  <sys:environment>
    <sys:moduleId>
      <sys:groupId>default</sys:groupId>
      <sys:artifactId>dwapp</sys:artifactId>
      <sys:version>1.0</sys:version>
      <sys:type>car</sys:type>
    </sys:moduleId>
    <sys:dependencies>
          <sys:dependency>
              <sys:groupId>console.dbpool</sys:groupId>
              <sys:artifactId>dwDatasource</sys:artifactId>
          </sys:dependency>
          <sys:dependency>
              <sys:groupId>dwr</sys:groupId>
              <sys:artifactId>dwr</sys:artifactId>
              <sys:version>1.1.4</sys:version>
          </sys:dependency>
    </sys:dependencies>
  </sys:environment>
...
```

下面的程序片段表示新的 DWR 配置文件 dwr.xml 添加到 WebContent\WEB-INF 目录中。dwr.xml 配置文件告诉 DWR 哪个 Java 类可以远程使用。它将 com.dw.*类作为 JavaScript 代码中的 dwrajax 类。<converter> 行指定 DWR 应允许在服务器和客户机之间传输 com.dw.*类型的对象。

```
<!DOCTYPE dwr PUBLIC
    "-//GetAhead Limited//DTD Direct Web Remoting 1.0//EN"
    "http://www.getahead.ltd.uk/dwr/dwr10.dtd">
<dwr>
  <allow>
    <convert converter="bean" match="com.dw.*"/>
    <create creator="new" javascript="dwrajax">
      <param name="class" value="com.test.dw*"/>
    </create>
  </allow>
</dwr>
```

6.3 Portlet

Portlets 在 Web 门户上管理和显示的可插拔的用户界面组件。Portlet 产生可以聚合到门户页面中的置标语言代码的片段，如 HTML，XML 等。通常一个门户页面显示为一组互相不重叠的 portlet 窗口，其中每一个 portlet 窗口显示一个 portlet。因此，可以说一个（或一组）portlet 就像一个在门户网站上运行的基于 Web 的应用程序。Portlet 应用程序的一些例子包括电子邮件、天气预报、论坛和新闻等。Portlet 标准的目的是使开发人员开发出的 portlet 可以插入到任何支持该标准的门户网站。而远程 Portlet 的 Web 服务（Web Services for Remote Portlets）协议的目的是提供 Web 服务标准，允许来自不同来源的远程 Portlet 可以"即插即用"。许多网站允许注册用户通过开关 Web 页面的某些部分或添加或删除特性，来自定制个性化的网站的面貌。这有时是通过共同构成该门户网站的一组 portlet 来完成的。

Java Portlet 规范（JSR-168，JSR-286）提供 portlet 在不同 Web 门户网站的互操作能力。该规范定义了 portlet 容器和 portlet 之间交互的一组 API 来解决个性化，展示和安全方面的问题。其中 Apache Pluto 是 JSR-168 的一个参考实现。除了参考实现，也有许多厂商提供了 portlet 容器的商业实现，一些主要的厂商如 IBM、Oracle、BEA、Vignette 和 SUN 等。这些厂商提供了基于 portlet 标准的实现，以及尚未被标准机构认可的扩展。此外，也有大量的开源 portal 解决方案支持 JSR-168，如 Apache 的 Jetspeed-2 Enterprise Portal、eXo Platform、uPortal、Liferay Portal 等。

JSR-168 目前在业界受到广泛支持，而且它由开放源码支持。该标准和产品的第一个版本存在一定的缺陷，仅支持最基本的用例，在功能上有一些限制。使得 Java Portlet Specification V1.0（JSR-168）也存在这种情况；因此，经过几年之后，大多数支持 Java Portlet Specification V1.0 的门户产品都提供一些附加扩展，以支持更高级的用例，这些附加的扩展造成了各个门户产品的标准不统一，彼此间的交互协作成了不可避免的问题。为了更好地规范 portlet 开发，以适应业界发展，并提供适应于最高级别用例的标准解决方案，从而为这些高级功能提供互操作性，于是在 2005 年 11 月开始了 Java Portlet Specification V2.0（称为 JSR-286）的开发，且 Java Portlet Specification V2.0 已在 2008 年正式发布。JSR-286 兼容了 JSR-168，并完善了 JSR-168 的部分功能，提供了诸多 JSR-168 所没有的新特性，例如资源服务、事件、portlet 过滤器、共享呈现参数及 portlet 窗口等。JSR-286 较 JSR-186 的新特性包括：

（1）资源服务：一种新的 portlet 呈现资源的方式。表示方式如下：

```
<a href="<c:url value="/test.jsp" />">test.jsp</a>//直接访问资源文件
<a href="<c:url value="/testServlet" />">testServlet</a>//直接访问 Servlet
```

JSR-286 引入了一个新的、具有 serveResource 方法的可选生命周期接口 ResourceServingPortlet，该接口可以由 ResourceURL 触发。Portlet 可以通过 PortletResponse.createResourceURL 方法创建它。

（2）事件：通过发送事件和接收事件来实现 portlet 之间的通信。

（3）Portlet 过滤器：与 servlet 过滤器类似，如图 6-5 所示，根据 Portlet 请求和响应动态的呈现内容的变换。存在以下四种类型的 portlet 过滤器：Action 过滤器、Render 过滤器、

Resource 过滤器、Event 过滤器。

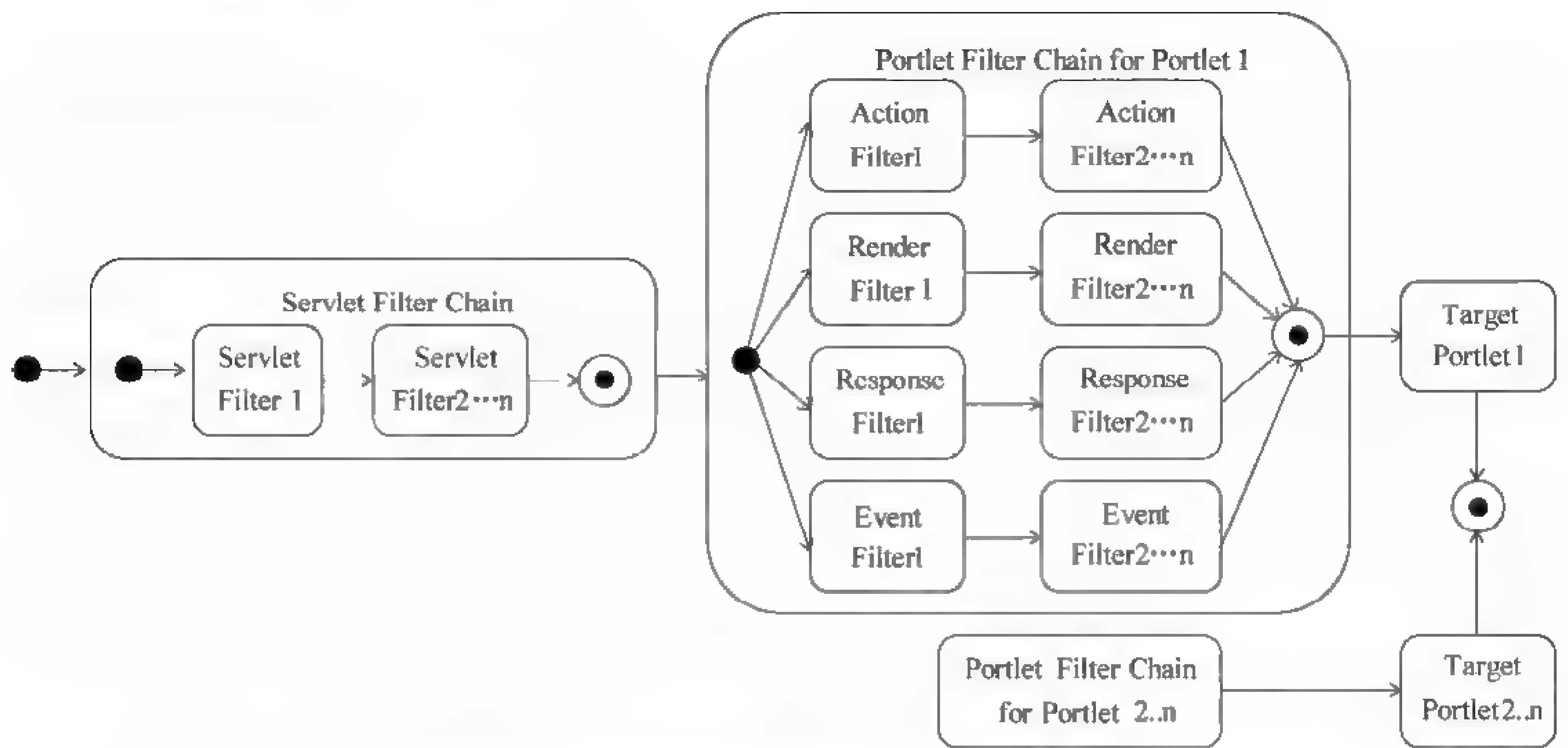

图 6-5　Servlet 过滤器链和 Portlet 过滤器链

Portlet 过滤器可以使用户改变一个 request 和修改一个 response。Filter 不是一个 Portlet，因此它不能产生一个 response，但能够在一个 request 到达 Portlet 之前预处理 request，也可以在离开 Portlet 时处理 response。它能实现的功能包括：在 Portlet 被调用之前截获，各过滤器与截获方法关系如表 6-1 所示；在 Portlet 被调用之前检查 servletrequest；根据需要修改 request 头和 request 数据；根据需要修改 response 头和 response 数据；在 Portlet 被调用之后截获。

表 6-1　过滤器与截获方法关系

过滤器	拦截方法
Action 过滤器	processAction(ActionRequest request, ActionResponse response)
Render 过滤器	render(RenderRequest request, RenderResponse response)
Resource 过滤器	serveResource(ResourceRequest request, ResourceResponse response)
Event 过滤器	processEvent(EventRequest request, EventResponse response)

JSR 286 为 Portlet 过滤器提供了接口类 PortletFilter，该接口提供了两个方法。

① void init(FilterConfig config)：容器调用一次这个方法来准备用于服务的过滤器。对象 filterConfig 使得过滤器能够访问配置参数，以及对门户上下文的引用。

② void destroy()：这个方法是在将过滤器从服务移除之后调用的。这个方法使得过滤器能够清除任何存放的资源。

并且 JSR-286 为四种过滤器分别定义了一个接口类，这四个接口类都继承 PortletFilter 类，并分别添加了各自 doFilter()方法，如图 6-6 所示。

在实现 Portlet 过滤器声明时，需要注意以下三点：对于一个 Portlet 过滤器的声明亦包括两部分，过滤器的定义声明和过滤器的映射声明；一个 Portlet 过滤器可以为多个 Portlet 服务，而且一个 Portlet 可以同时有多个 Portlet 过滤器；一个 Portlet 过滤器可以有多个生命周期阶段，当然前提是该 Portlet 过滤器实现了相应过滤器接口。例如下程序片段：

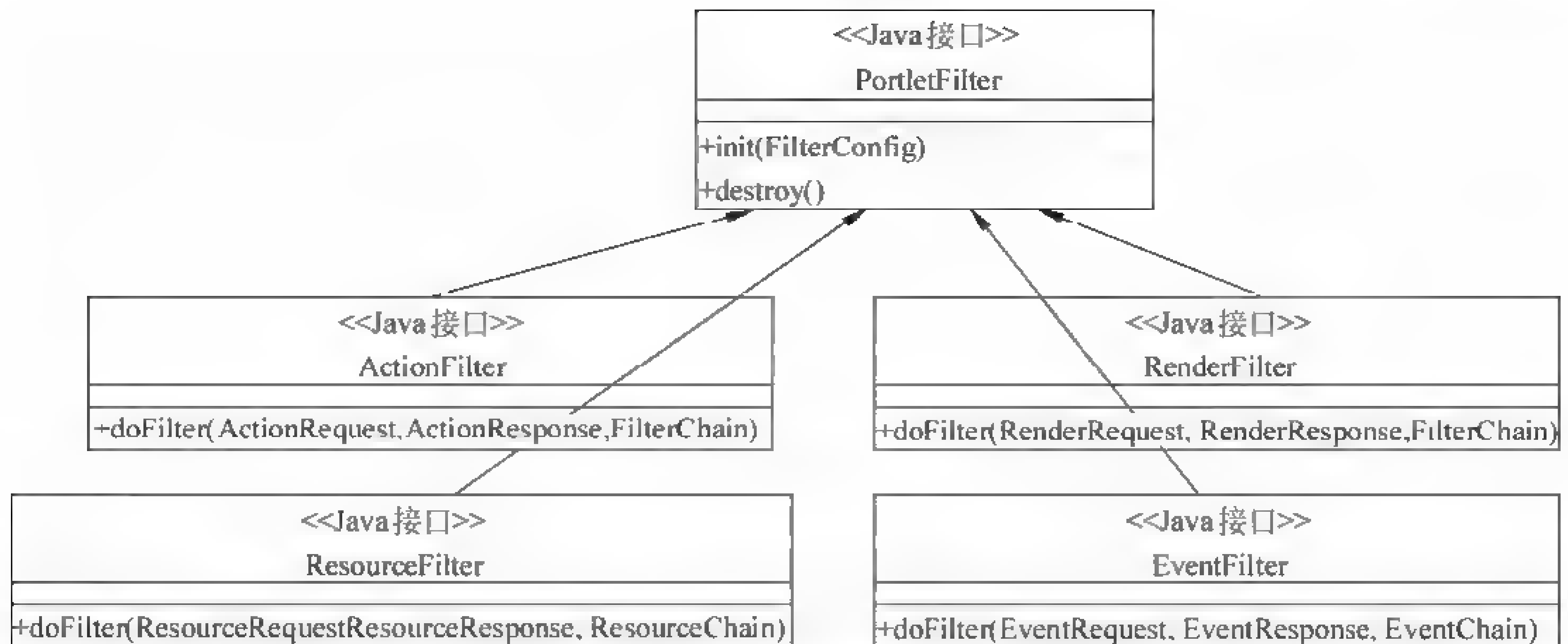

图 6-6　Portlet 过滤器继承图

```
<portlet-app ...>
    ...
    <filter>
        <filter-name>TestRenderFilter</filter-name>
        <filter-class>com.jsr286.TestRenderFilter</filter-class>
        <lifecycle>RENDER PHASE</lifecycle>
    </filter>
    <filter>
        <filter-name>TestAllFilter</filter-name>
        <filter-class>com.jsr286.TestAllFilter</filter-class>
        <lifecycle>ACTION PHASE</lifecycle>
        <lifecycle>RENDER PHASE</lifecycle>
        <lifecycle>EVENT PHASE</lifecycle>
        <lifecycle>RESOURCE PHASE</lifecycle>
    </filter>
    <filter-mapping>
        <filter-name>TestRenderFilter</filter-name>
        <portlet-name>DocumentPortlet</portlet-name>
        </filter-mapping>
    <filter-mapping>
        <filter-name>TestAllFilter</filter-name>
        <portlet-name>Test*</portlet-name>
    </filter-mapping>
     ......
</portlet-app>
```

（4）共享呈现参数：除了 portlet 私有的呈现参数之外，新增了可以在 portlet 之间共享的呈现参数。

（5）Portlet 窗口：提供 portlet 窗口 ID 供 portlet 使用。

开源软件 Apache Pluto 是 Apache Portals 项目的子项目，其结构如图 6-7 所示。它是 Java Portlet Specification 的开源实现。Pluto 项目提供了符合规范要求的 portlet 容器运行时环境，可以在其中初始化和管理 portlet。它最初是通过 Java Community Process 和 Java Specification

Request (JSR) 168 创建的。该规范定义了用 Java 编程语言开发门户和 portlet 组件的指导信息，也被视为传统门户的通用标准和构建可移植 Web 应用程序的框架。

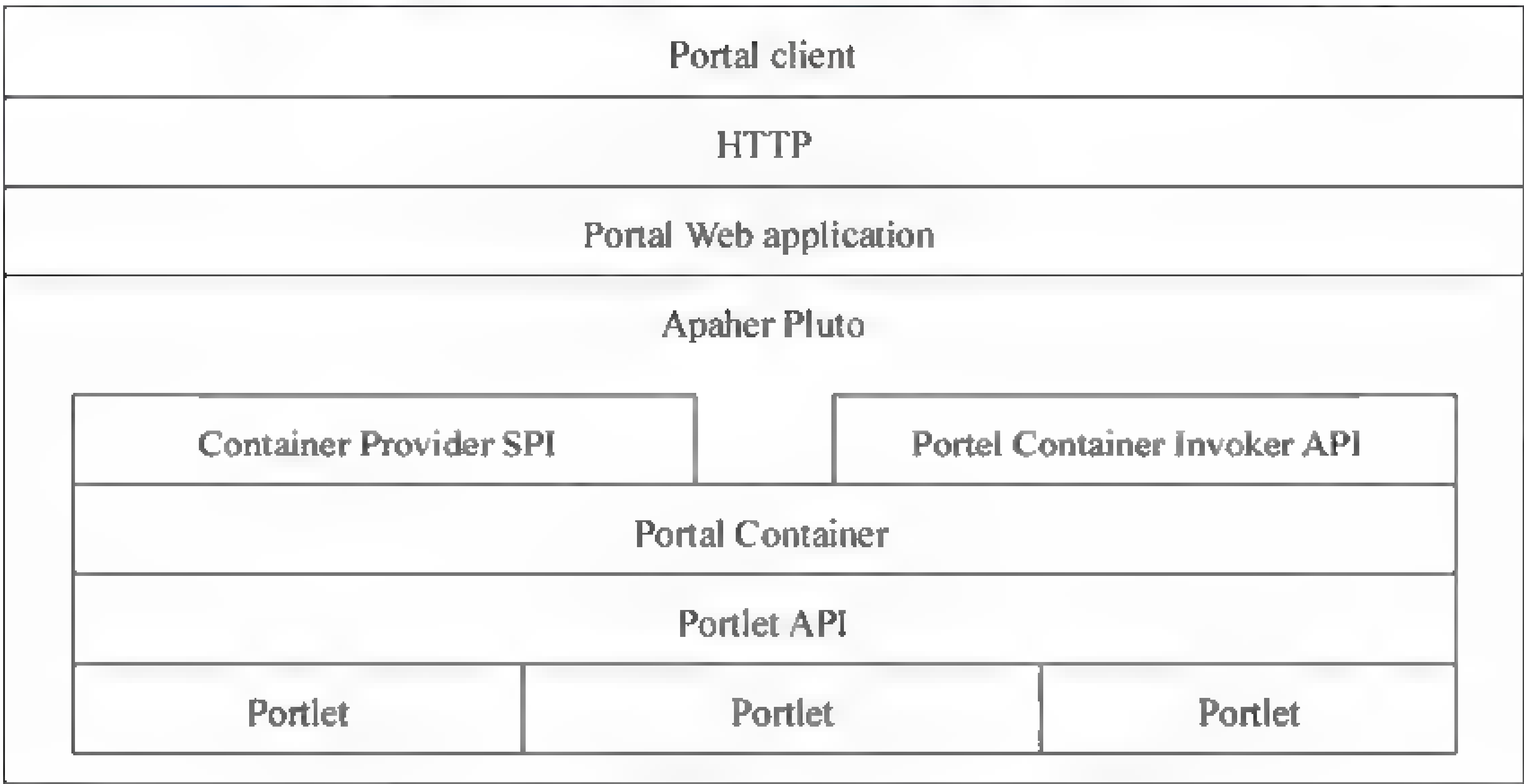

图 6-7　Apache Pluto 结构

6.3.1　容器

容器可以管理对象的生命周期、对象与对象之间的依赖关系，可以使用一个配置文件（通常是 XML）来配置在容器上面定义好对象的名称、如何产生（Prototype 方式或 Singleton 方式）、哪个对象产生之后必须设定成为某个对象的属性等。这样，在启动容器之后，所有的对象都可以直接取用，不用编写任何一行程序代码来产生对象，或是建立对象与对象之间的依赖关系。容器是一个 Java 所编写的程序。Java 容器类通常包含 List、ArrayList、Vector 及 map、HashTable、HashMap。

大多数容器 API（如 EJB API）强迫程序员编写一些接口或一个组件模型。并将组件放入该容器后，容器会处理一些事情。EJB 容器提供企业服务，Servlet 容器（例如 Apache Jakarta Tomcat）实现了 Servlet API，从而使动态内容建立到服务器页面中，该页面随后会被发送到 Web 浏览器。同时，传统容器强迫使用指定的编程模型，而轻量级容器则不是。它们使用普通 Java 对象（plain old Java object，POJO）。并且轻量级容器有许多胜于其他容器架构的优点。例如，程序员可以使用一个更加简单、基于 POJO 的编程模型。这样应用程序会更加易于测试。对象也可以在容器外运行。如在一个测试用例中，通过依赖注入，轻量级容器减少了组件间的依赖性。

Portlet 在 Portlet 容器中运行，Portlet 容器为 Portlet 提供必需的运行环境。Portlet 容器包含 Portlet（组件）并且管理它们的生命周期，它也为 Portlet 的参数设置提供持久化的存储。Portlet 容器不是一个类似于 servlet 容器的独立容器。它是在 servlet 容器上通过扩展方式实现的，并重用 servlet 容器提供的功能。从 Portal 的角度来看，Portlet Container 是 Portal 平台所提供的众多服务之一。但是 Portal 并不等价于 Portlet Container。一个企业级的门户实现，可以包含或者说支持多种 Portlet Container 同时运行，对于 Jetspeed 而言，它同样关注的是门户本身的实现，而不是 Portlet Container 的实现，但由于 Jetspeed-2 完全抛弃 Jetspeed-1 的架构，

而决定彻底拥抱标准，因此 Jetspeed 只有一个标准的 Portlet Container 实现，这就是 Pluto 项目。

而对于 Portal 就是一个大的容器，容器中有多个由 Portlet 来填充内容以展现给用户，每个 Portlet 在容器中就相当于一个相对独立的 web 组件，它们可以单独的向 web 服务器发出 http 请求以更新自己的 Portlet 中的内容，如图 6-8 所示。

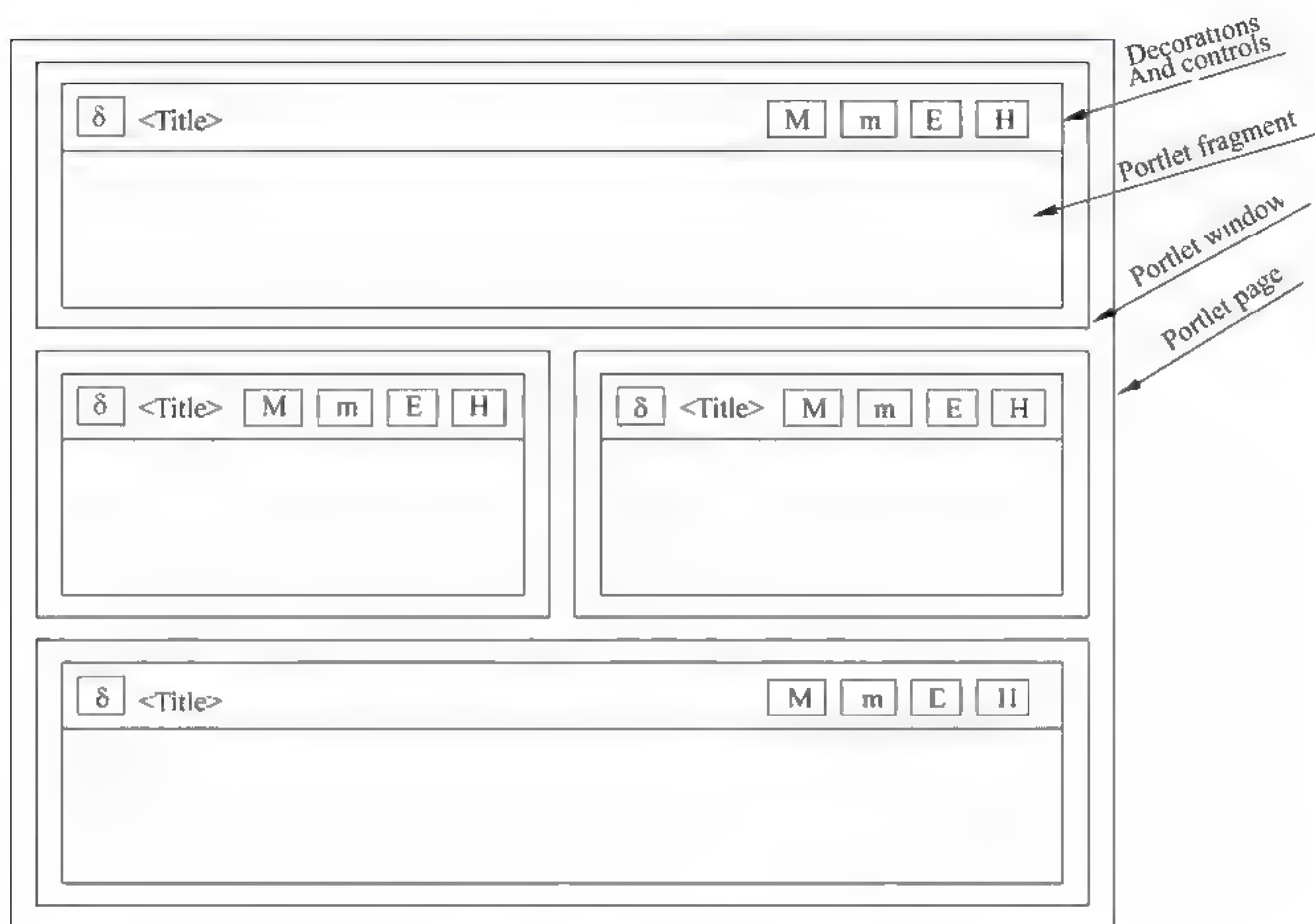

图 6-8　Portlet 页面容器结构

Portlet 是基于 java 技术的 web 组件，它由 portlet 容器管理、并处理请求，并动态生成输出内容。就像 servlets 是专为将合成页面里的内容聚集在一起而设计的。通常请求一个 portal 页面会引发多个 portlets 被调用。每个 portlet 都会生成标记段，并与别的 portlets 生成的标记段组合在一起嵌入到 portal 页面的标记内。并且每个 portlet 页面由一个或多个 portlet 窗口组成，每个 portlet 窗口又分为两部分：一个是主题外观（Theme），它决定了 portlet 窗口的标题条、控制和边界的样式；另一个是 portlet 段，它由 portlet 应用填充。也就说明 Portlet 是基于 Java 平台的 Web 门户应用程序。JSR-168 是开发 portlet 应用程序的 Java Community Process 标准，它描述了 portlet 生命周期管理、portlet 容器合约、打包、部署以及与门户有关的其他方面。

6.3.2　页面处理

在 Portal 的开发过程中，Theme（主题外观）与 portlet 之间的通信，以及 portlet 之间的通信是开发人员常常遇到的问题。通常 Portlet 之间需要能互相通信，即一个 portlet 的状态发生改变，要通知其他的 portlet，这些收到通知的 Portlet 状态也要做相应的改变。根据 JSR168

规范，Portlet 由容器管理，它们之间是相互独立的，并不共享 Session 对象。

当在同一页面内的不同 portlet 间的通信时，通常对放在同一个页面上的 Portlets，如果其中的某个 Portlet 做了提交等操作导致 portal 刷新，这个页面内的每一个 portlet 都会被 render。在 Portlet 中，要实现参数的传递通常有三个步骤：①定义源 portlet，②定义目标 portlet，③关联源 portlet 和目标 portlet。

当在 Ajax 与 portlet 间通信时，似乎 Portlet 和 Ajax 看起来彼此之间是完美搭配，因为它们都侧重于用 Web 浏览器作为向用户呈现用户界面的工具。但把这两者与 Java 技术组合在一起的简易方式就是使用 DWR 库。在实现 AJAX 与 Portlet 通信时，首先需要设置 web.xml 和 dwr.xml（具体设置见前一节 DWR 内容）。其次将具有 AJAX 的 JSP 导入了来自 DWR 的 JavaScript 库（engine.js），并动态地创建库相应的业务 JS 文件；并使用的 JS 名称与 dwr.xml 中的 JavaBean 的名称相同即可。最后创建 Portlet 完成与 AJAX 通信。这里就不详细编入具体实现流程[1]。

下面继续介绍在 Apache Geronimo 和 Eclipse 中部署和构建 portlet 的流程。

6.3.2.1 配置 web.xml 和 geronimo-web.xml。

web.xml 是用来描述 Web 组件的，而 geronimo-web.xml 将告诉 Geronimo 应当如何打包和部署应用程序。在项目中，修改 web.xml 使其内容与如下程序片段中所示的内容相匹配，并以 LLiferay[2]门户为例进行介绍。

web.xml 配置：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp ID" version="2.4" xmlns="http://java.sun.com/xml/ns/
j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app 2 4.xsd">
   <display-name>simpleportlet</display-name>
   <context-param>
                  <param-name>company id</param-name>
                  <param-value>Liferay.com</param-value>
                  </context-param>
                  <listener>
                  <listener-class>
                  com.Liferay.portal.kernel.servlet.PortletContextListener
                  </listener-class>
                  </listener>
                  <servlet>
                  <servlet-name>simpleportlet</servlet-name>
                  <servlet-class>
                  com.Liferay.portal.kernel.servlet.PortletServlet
```

① http://www.ibm.com/developerworks/cn/java/j-ajaxportlet/

② http://www.Liferay.com/

```
                    </servlet-class>
                    <init-param>
                    <param-name>portlet-class</param-name>
                    <param-value>org.dworks.SimplePortlet</param-value>
                    </init-param>
                    <load-on-startup>0</load-on-startup>
                    </servlet>
                    <servlet-mapping>
                    <servlet-name>simpleportlet</servlet-name>
                    <url-pattern>/simpleportlet/*</url-pattern>
                    </servlet-mapping>
</web-app>
```

配置 geronimo-web.xml，实现 Portlet 部署：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.1"
xmlns:nam="http://geronimo.apache.org/xml/ns/naming-1.1"
xmlns:sec="http://geronimo.apache.org/xml/ns/security-1.1"
xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.1">
   <sys:environment>
      <sys:moduleId>
         <sys:groupId>default</sys:groupId>
         <sys:artifactId>simpleportlet</sys:artifactId>
         <sys:version>1.0</sys:version>
         <sys:type>car</sys:type>
      </sys:moduleId>
      <sys:dependencies>
                    <sys:dependency>
                    <sys:groupId>Liferay</sys:groupId>
                    <sys:artifactId>Liferay-portal-tomcat</sys:artifactId>
                    </sys:dependency>
                    </sys:dependencies>
   </sys:environment>
</web-app>
```

在配置文件中，<moduleId>元素将包含四个子元素，用于指定项目的已部署模块应当放在 Geronimo 资源库中的位置：groupId[创建文件的实体（例如 Apache）]、artifactId（文件名）、version（文件版本）、type[文件的格式（例如 JAR）]。

6.3.2.2　在 Liferay 中创建 portlet.xml

web.xml 和 geronimo-web.xml 文件足以部署 servlet 或基于 JSP 的应用程序，但是需要将第三个部署描述符 portlet.xml 用于 portlet 应用程序。然后，由于此 portlet 必须被集成到 Liferay（可以在 http://sourceforge.net/projects/lportal/files/中下载）门户中。因此需要另外两个文件：Liferay-portlet.xml 和 Liferay-display.xml。

Liferay 作为一个开源的 Portal 项目，利用 Hibernate、Struts、Spring 等开源框架，实现

了 JCP JSR-168 规范中提出的 Protal 功能，在开源 Protal 系统中比较典型的代表性。它代表了完整的 J2EE 应用，使用了 Web、EJB 及 JMS 等技术，特别是其前台界面部分使用 Struts 框架技术，基于 XML 的 portlet 配置文件可以自由地动态扩展；并使用了 Web Services 来支持一些远程信息的获取，使用 Apache Lucene 实现全文检索功能。主要提供了以下功能：

（1）提供单一登录接口，多认证模式（LDAP 或 SQL）；

（2）管理员能通过用户界面轻松管理用户、组、角色；

（3）用户能可以根据需要定制个性化的 portal layout；

（4）能够在主流的 J2EE 应用服务器上运行，如 JBoss+Jetty/Tomcat、JOnAS；

（5）支持主流的数据库，如 PostgreSQL、MySQL；

（6）使用了第三方的开源项目，如 Hibernate、Lucene、Struts；

（7）支持包括中文在内的多种语言；

（8）采用最先进的技术，如 Java、EJB、JMS、SOAP、XML。

目前，Liferay 分为 Professional 和 Enterprise 两个版本。Liferay 支持多个应用服务器和 Servlet 容器，而 Liferay Enterprise 版本需要一个健壮的 J2EE 服务器，而 Liferay Professional 版本只要一个普通的 Servlet 服务器就可以运行。如果需要运行 EJB 通常使用 Professional 版本，但两个版本的源代码和应用接口都是一样的。在默认情况下，Professional 版本分别集成 Tocmcat/Jetty/Resin 作为 Web 服务器，采用 Struts 作为 Web 框架，实现轻量级的系统架构。而 Enterprise 版本需要集成 JBoss 做为 Web 服务器，采用 Spring 作为 Web 框架，并兼顾 EJB。

Liferay 默认集成 HSQL 数据库，来持久化保存用户自定义的数据。通过修改集成在 Liferay 的 Tomcat 的部署描述文件，当然用户也可以根据需要更改数据源。在 Liferay 官方网站中也提供了数据库的生成脚本。下面继续介绍 Liferay 安装步骤：

（1）在 http://sourceforge.net/projects/lportal/files/下下载 Tomcat 版，也可以下载 Jboss、Jetty 等版本。

（2）下载并安装 JDK 1.5 以上版本，这是别忘了设置“环境变量”。

（3）在 Liferay 安装目录下启动 startup.bat，然后自动启动 http://localhost:8080/，如图 6-9 所示。

图 6-9　Liferay 启动页面

（4）在图 6-9 右上角点击"Sign In"，可以采用用户名和密码都是 test 进行登录，并且成功登录后可以改成中文界面。当 lLiferay 自动启动后，HSQL 数据库会自动启动，如图 6-10 所示登录界面，登录成功的界面如图 6-11 所示。

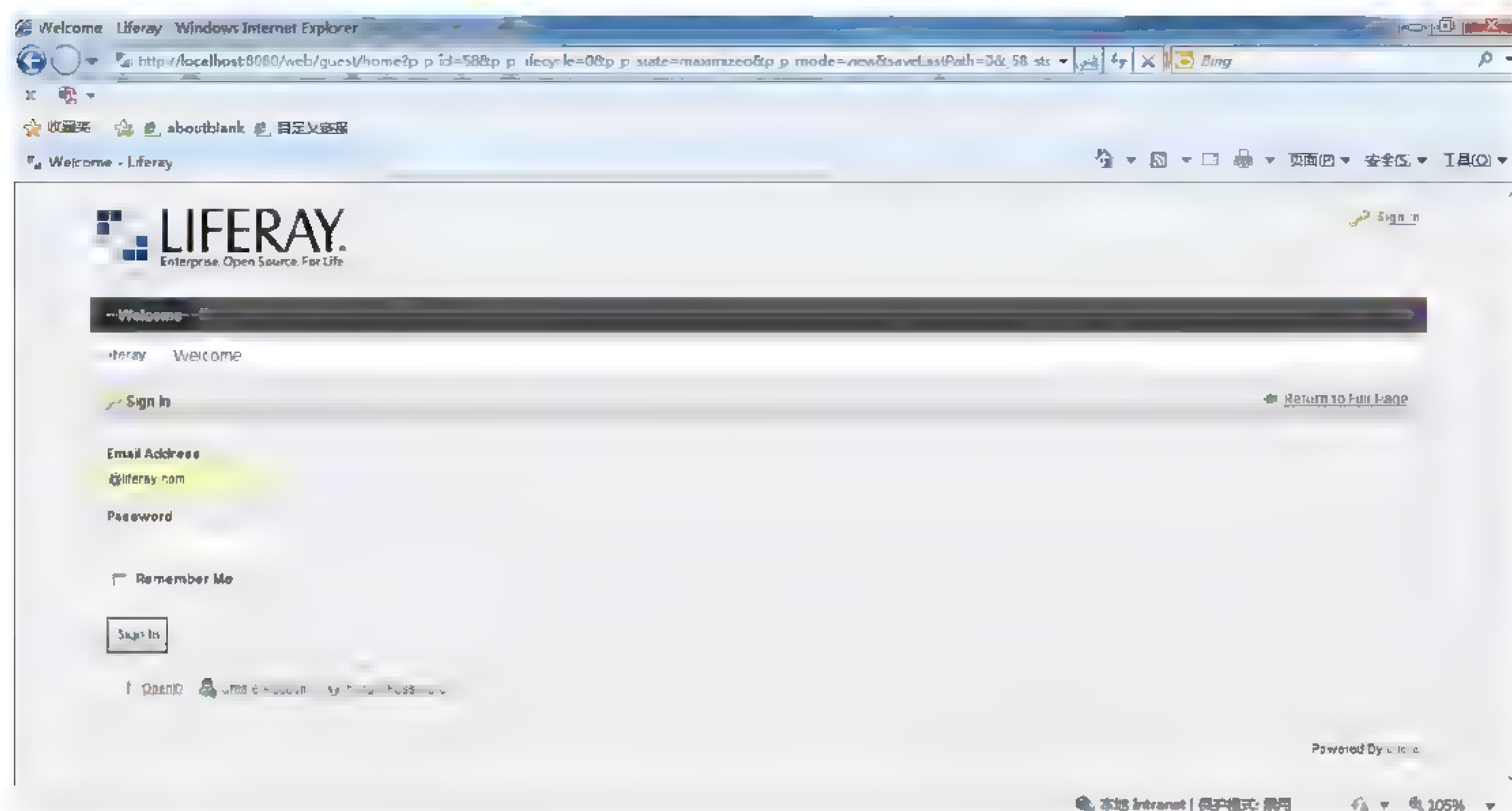

图 6-10　Liferay 的登录界面

图 6-11　Liferay 登录成功后的界面

下面进行 portlet 应用的配置文件设置：

（1）配置 portlet.xml：

```
<?xml version="1.0"?>
```

```
<portlet app xmlns "http://java.sun.com/xml/ns/portlet/portlet-app 1 0.xsd"
version "1.0" xmlns:xsi "http://www.w3.org/2001/XMLSchema instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app 1 0.xsd
http://java.sun.com/xml/ns/portlet/portlet-app 1 0.xsd">
   <portlet>
      <portlet-name>simpleportlet</portlet-name>
      <display-name>Simple Portlet</display-name>
      <portlet-class>org.dworks.SimplePortlet</portlet-class>
      <expiration-cache>0</expiration-cache>
      <supports>
         <mime-type>text/html</mime-type>
      </supports>
      <portlet-info>
         <title>Simple Portlet</title>
         <short-title>Simple Portlet</short-title>
         <keywords>Simple Portlet</keywords>
      </portlet-info>
      <security-role-ref>
         <role-name>administrator</role-name>
      </security-role-ref>
      <security-role-ref>
         <role-name>guest</role-name>
      </security-role-ref>
      <security-role-ref>
         <role-name>power-user</role-name>
      </security-role-ref>
      <security-role-ref>
         <role-name>user</role-name>
      </security-role-ref>
   </portlet>
</portlet-app>
```

（2）配置 lLiferay-portlet.xml：

```
<?xml version="1.0"?>
<!DOCTYPE Liferay-portlet-app PUBLIC "-//Liferay//DTD Portlet Application
4.2.0//EN"
"http://www.Liferay.com/dtd/Liferay-portlet-app 4 2 0.dtd">
<Liferay-portlet-app>
   <portlet>
      <portlet-name>simpleportlet</portlet-name>
      <instanceable>true</instanceable>
   </portlet>
   <role-mapper>
      <role-name>administrator</role-name>
      <role-link>Administrator</role-link>
```

```
    </role mapper>
    <role mapper>
      <role name>guest</role name>
      <role-link>Guest</role-link>
    </role-mapper>
    <role-mapper>
      <role-name>power-user</role-name>
      <role-link>Power User</role-link>
    </role-mapper>
    <role-mapper>
      <role-name>user</role-name>
      <role-link>User</role-link>
    </role-mapper>
</Liferay-portlet-app>
```

在配置文件中，需要将把 portlet.xml 的安全角色与 Liferay 中特定并类似命名的角色相匹配。如果 instanceable 元素被设为 true，则 portlet 可被多次添加到 Liferay 门户的布局中。

（3）配置 lLiferay-display.xml：

```
<?xml version="1.0"?>
<!DOCTYPE display PUBLIC "-//Liferay//DTD Display 4.0.0//EN"
"http://www.Liferay.com/dtd/Liferay-display 4 0 0.dtd">
<display>
    <category name="category.test">
      <portlet id="simpleportlet" />
    </category>
</display>
```

在配置文件中，将告诉门户 portlet 将列于哪个类别中。在说明如何部署和查看 Liferay portlet 的过程中，在 Liferay 中可以设置类别。

这时，创建完了五个描述符（web.xml、geronimo-web.xml、portlet.xml、Liferay-portlet.xml、Liferay-display.xml）后，就已经把服务器与门户连接了起来，实现 portlet 页面间通信。

6.3.3　Jetspeed

Jetspeed[①]是 Apache 组织开发的一个采用 Java 和 XML 的开放源代码的企业信息门户的实现。门户可以让终端用户很方便的访问网络资源（应用、数据库等）。用户可以通过 Web 浏览器、WAP 手机、寻呼机以及其他一些智能设备来访问 Portal。Jetspeed 就像是中心的控制器，可以很方便地以各种形式展示那些来自不同数据源的数据。并且通过 Jetspeed 展示的数据形式完全独立于内容的类型。这就意味着 Jetspeed 可以集成各种各样的数据源，如：XML、RSS、SMTP。然后通过 XSL 技术将数据组织成 Jsp 页面或 Html 页面传给客户端。Jetspeed 还支持模板和内容的发布框架，比如 Cocoon、WebMacro、Velocity。图 6-12 所示是 Jetspeed

① http://portals.apache.org/jetspeed-2

执行流程。

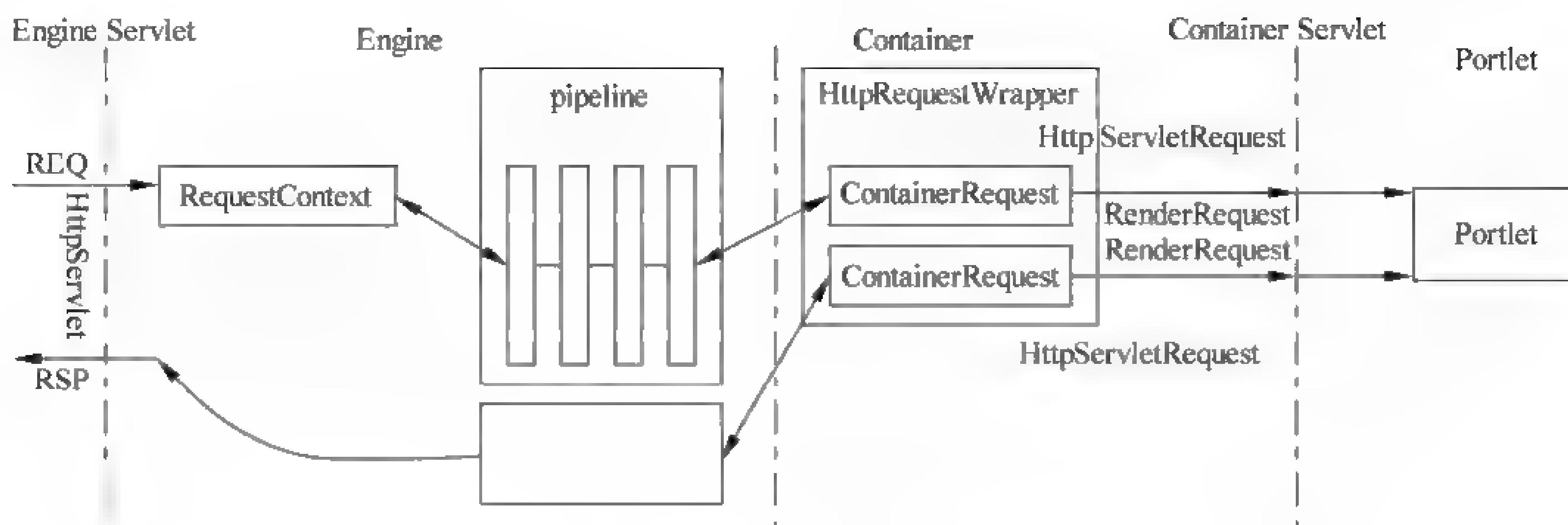

图 6-12　Jetspeed 执行流程

6.3.3.1　Jetspeed 概述

Jetspeed 采用 Turbine[①]作为主要的框架支持，Turbine 为 Jetspeed 提供用户认证、页面布局管理和计划服务等。使得 Portlet 可以直接使用 Turbine 服务提供的 RunData 对象。Jetspeed 向用户展示的页面由 Turbine 控制产生，它的主要内容由一些标准的 Portlet 构成。Portlet 设计的目标是[②]：

（1）一个页面上可以提供很多小的 WEB 应用程序给用户。

（2）这些 WEB 应用程序的背景色、标题栏颜色、图标都可以更换；同时，Portlet 允许对它的显示风格进行定制，比如背景色、尺寸等。

（3）可以用缓冲子系统维持跨多个 Portlet 的应用。

（4）可以对所有的 WEB 应用程序进行管理、维护，并提供给用户。

（5）简单的选择就可以让用户定制页面，这些页面除了一个页面，还可以显示多个 Portlet，这和 Turbine 的显示页面很相识。

（6）由于缓冲子系统的存在，使得系统可以快速运行。这使得即使要和数据库打交道得复杂 Portlet 也可以快速生成 Html 页面。

（7）很容易开发整个系统，开发人员不必知道整个 Jetspeed 的运行机制。

（8）Portlet 可以通过多种方式形成页面。可以通过 JDBC 从数据库记录产生页面；也可以通过 XML—>XSL—>HTML 产生页面；还可以通过其他途径如 CoCoon 形成页面。

（9） Portlet 通过 PortletController 来管理。这个 PortletController 是标准接口的实现，开发者可以定制它。

（10）Portlet 交给 PortletControl 来处理，PortletControl 给 Portlet 加上显示风格后返回 Portlet 的内容。

（11）Jetspeed 内通过 XML 标记文件分类管理 Portlet。

（12）Portlet 接受一个 PortletConfig 参数，这个参数中包括 Url 地址和一些参数的哈希表。

（13）大多数简单的 Portlet 可以通过继承 AbstractPortlet 来实现。

① http://turbine.apache.org

② http://www.oschina.net/p/jetspeed

根据 Jetspeed 的特征及应用结构可以将其分为 9 类来进一步了解，如表 6-2 所示。

表 6-2　Jetspeed 组成

Jetspeed 特征类别	描述
标准	完整兼容 JSR-168，目前的版本尚不支持 JSR-268
	通过 JSR-168 规范兼容性测试
	基于 JAAS（Java Authentication And Authorization Security）标准的认证和授权服务（默认支持数据库的实现）
	基于 LDAP（Lightweight Directory Access Protocol）的用户认证
体系构架	基于 Spring Framework 的组件架构
	灵活可配置的请求通道（通过 Spring Bean XML 配置）
	Portlet 应用发布单元热部署
	Jetspeed AJAX XML API（基于著名的开源 AJAX Framework-DOJO）
	扩展的 Portlet 页面结构语言（支持持久化到文件或数据库）
门户核心特性	声明风格的安全约束
	基于角色的 Portlet 安全方面的 API
	门户内容管理和导航，包括页面、菜单、文件夹和超链接
	单线程或多线程的内容聚合引擎（通过 Spring Bean 可以轻易切换）
	高度可扩展的 Jetspeed 单点登录服务框架
	基于权限和规则的门户页面和资源定位配置
	支持所有主流的数据库，包括：Derby、MySQL、MS SQL、Postgres、Oracle、DB2、Hypersonic
	不依赖客户端类型的 capability engine（html, xhtml, wml,vml）
	多语言支持（12 国语言，包括简体中文和繁体中文），而且完全可扩展完整的性能统计日志引擎
	利用著名开源搜索引擎 Lucene 提供对所有门户资源的全文本检索和元数据搜索服务
	用户注册服务和忘记密码的邮件通知服务
	丰富的登录密码配置策略
门户管理	用户，角色，用户组，密码和 Profile 管理
	JSR 168 协议规范定义的用户属性编辑器
	门户页面管理
	单点登录服务管理
	Portlet 应用程序管理
	Profiler 管理
	门户性能统计报告
对 Web 框架的支持和例子 Portlets	通过 Bridges 项目支持几乎所有的主流 Web Framework 与 Jetspeed 门户的整合，包括：JSF（Sun 的标准 JSF 实现和 Apache MyFaces）、Apache Struts、PHP、Perl、Velocity
	例子 Portlet 包括：RSS、IFrame（通过 Jetspeed SSO API 还可以支持 SSO 效果）、日历、书签
	支持 Spring MVC

续表

Jetspeed 特征类别	描述
用户个性化	门户页面管理
	页面用户定制（包括增删查改门户页面，页面的风格，Portlet 框体风格，Portlet 的位置，Portlet 的布局等等）
	支持两种门户定制风格，包括传统的基于页面刷新的风格和基于 AJAX 技术的风格
门户设计	支持 Portlet 和 Portal 页面皮肤的打包发布
	基于 CSS 技术的可配置布局
	支持 Velocity 模版引擎
门户开发工具	支持 Maven 1.x 和 Maven2.0.x，部分功能支持 Ant 脚本
	支持通过 Maven 插件生成自定义门户基础框架
	热部署 Portlet 应用发布单元和门户资源
	支持通过 API 调用的方式部署 Portlet 应用发布单元
	支持 Eclipse3.2.x 开发环境
应用服务器	Apache Tomcat
	JBoss
	Geronimo

表 6-3 所示是 Jetspeed 的核心组件描述。

表 6-3　Jetspeed 的核心组件描述

组件名	路径	描述
Jetspeed-Api	components/jetspeed-api	定义几乎所有的 jetspeed-api interfaces，一般的开发者都使用这个组件中定义的接口进行二次开发
Component Manager	components/cm	通过接口 org.apache.jetspeed.components. ComponentManager 屏蔽了 Spring 的实现细节。可以通过实现该接口替换 Spring
Deploy-Tool	components/deploy-tool	当 Web Container 为 Tomcat 时，通过该组件，读取已打包好的 portlet 应用程序中的 portlet.xml 和 web.xml，检查是否包含 JetspeedContainerServlet 的定义，如果没有则修改 web.xml 加入这部分信息
Id-Generator	components/id-generator	用于生成全局唯一的 portlet 实例 id
Locator	components/locator	提供定位门户资源的服务，资源包括：模板，Profiler 等
Page-Manager	components/page-manager	对著名的门户结构描述文件-PSML（Portal Structure Markup Language），提供了 Java 对象模型映射，并且支持文本风格的 PSML 和数据库风格的 PSML，以及 PSML 管理器
Portal	components/portal	实现绝大部分的 jetspeed-api 组件中定义的 interface，是最核心的组件
Preferences	components/prefs	实现了 Portlet 属性偏好功能，提供将这些属性持久化到数据库的服务
RDBMS	components/rdbms	Jetspeed 中所有与 Apache OJB O/R Mapping 框架有关的组建的基础组件
Search	components/search	提供整个门户资源的全文本搜索服务，具体实现依赖于 Apache Lucene

续表

组件名	路径	描述
Security	components/security	提供基于标准 JAAS 的认证服务，支持数据库和 LDAP 作为认证信息仓库。基于角色的授权服务，默认支持数据库作为权限仓库
Single Sign-on	components/sso	提供一个可扩展的单点登录服务接口和一个简单的基于 JAAS Subject 的实现，该组件主要提供 Portal 门户与后台应用之间的单点登录功能
Statistics	components/statistics	提供一个简单的访问请求统计服务的实现，支持将统计信息持久化到数据库。在 Jetspeed-2 管理界面中，还提供了专门的 Portlet 浏览这些统计信息

6.3.3.2　Jetspeed 与 SpringMVC 集成

Jetspeed 架构最大特点是其高度可定制性，并能选用著名开源 POJO 框架 Spring 作为其底层实现。在项目之初，Jetspeed 的开发者们也面临着 Spring 和 PicoContainer 的抉择，但事实证明当初的选择是正确的，因为随着 Spring 不断成长完善，Jetspeed 的组件架构也跟着收益良多。从另一个角度来看，Jetspeed 也可以作为利用 Spring 构建自己产品架构的经典范例[①]，如图 6-13 所示。

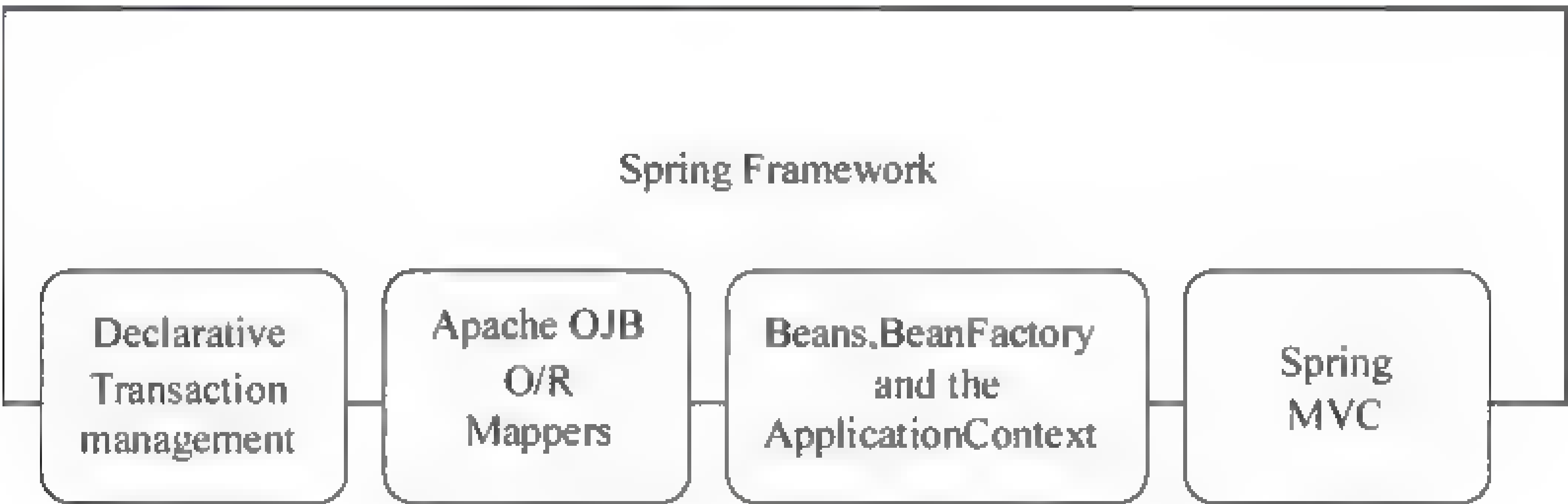

图 6-13　Jetspeed 使用到的 Spring 组件

Jetspeed 主要使用了 Spring 最核心 IoC 引擎中的 BeanFactory 和 ApplicationContext，来管理所有 Jetspeed Components 的生命周期和依赖关系，所有这些组件的 Spring 声明全部定义在名为 assembly 文件夹中的 XML 文件里。如果第三方开发者认为默认的 Jetspeed 组件不足以满足要求，只要按照自己需求编写 Jetspeed Component 的 Interface 的实现类，然后修改 Spring Bean XML 定义，就可以轻易替换掉默认的实现，例如下面配置程序片段如下：

```
<beans default-lazy-init="false" default-dependency-check="none"
default-autowire="no">
 <!--
SEARCH COMPONENT
  -->
```

① http://www.ibm.com/developerworks/cn/opensource/os-apache-portal/index.html

```
<bean id "org.apache.jetspeed.search.SearchEngine"
class "org.apache.jetspeed.search.lucene.SearchEngineImpl" abstract-
"false"
singleton="true" lazy-init="default" autowire="default"
dependency-check="default">
       <constructor-arg index="0">
          <value>${applicationRoot}/WEB-INF/search index</value>
       </constructor-arg>
       <constructor-arg index="1">
          <null />
       </constructor-arg>
<constructor-arg type="boolean">
          <value>true</value>
       </constructor-arg>
       <constructor-arg>
          <ref bean="org.apache.jetspeed.search.HandlerFactory" />
      </constructor-arg>
</bean>
</beans>
```

Jetspeed 在实现过程中遵循着面向接口编程的最佳实践，在图 6-13 中的 Bean 为 org.apache.jetspeed.search.SearchEngine，事实上这是一个定义在核心 jetspeed-api 组件中的接口，org.apache.jetspeed.search.lucene.SearchEngineImpl 为该接口的实现类，这个类定义在 components/search 组件中，后面的内容就是 SearchEngineImpl 的构造函数的输入参数，而最后一个参数 org.apache.jetspeed.search.HandlerFactory 也是一个 Java Interface 的接口。Spring 在实例化 SearchEngine 的时候，会首先分析它的构造函数参数是否已经全部满足条件（实例化），Spring 会根据搜索 bean id 为 org.apache.jetspeed.search.HandlerFactory 的 bean，如果已经实例化就直接注入到 SearchEngineImpl 的构造函数调用里；如果没有就实例化这个 bean 之后，再注入。由图 6-13 易知，通过 Spring 的 Declarative Transaction 机制，Jetspeed 很轻易实现细颗粒度的事物管理，用户可以很容易配置需要管理事务的方法，如 addSite*、updateSite*、removeSite 等等。

同时，由于 Spring 对 Apache OJB 提供良好的支持，因此 Jetspeed 中与数据库相关的功能基本上都用 Spring 的 PersistenceBrokerDaoSupport 来实现。这些组件包括：Capablity、DatabasePageManager、PipeLine、Preferences、Profiler、Registry、Security、SSO 等。O/R Mapping 的信息定义在上面这些组件 jar 包中的 JETSPEED-INF/ojb/component name repository.xml 文件中，其中 component name 需要用组建名称替代。下面一段 XML 定义了 SSOProvider 的事物管理：

```
<beans default-lazy-init-"false" default-dependency check-"none"
default autowire-"no">
```

```
<!-
 SSO Implementation
  -->
<bean id-"PersistenceBrokerSSOProvider"
class="org.apache.jetspeed.sso.impl.PersistenceBrokerSSOProvider"
init-method="init" abstract="false" singleton="true" lazy-init="default"
autowire="default" dependency-check="default">
       <constructor-arg index="0">
          <value>JETSPEED-INF/ojb/sso repository.xml</value>
      </constructor-arg>
   </bean>
<bean id="org.apache.jetspeed.sso.SSOProvider" parent="baseTransactionProxy"
name="ssoProvider" abstract="false" singleton="true" lazy-init="default"
autowire="default" dependency-check="default">
       <property name="proxyInterfaces">
          <value>org.apache.jetspeed.sso.SSOProvider</value>
       </property>
<property name="target">
          <ref bean="PersistenceBrokerSSOProvider" />
       </property>
       <property name="transactionAttributes">
<props>
             <prop key="addSite*">PROPAGATION_REQUIRED</prop>
            <prop key="updateSite*">PROPAGATION_REQUIRED</prop>
            <prop key="removeSite">PROPAGATION REQUIRED</prop>
  <prop key="addCredentialsForSite">PROPAGATION REQUIRED</prop>
<prop key="updateCredentialsForSite">PROPAGATION REQUIRED</prop>
<prop key="removeCredentialsForSite">PROPAGATION_REQUIRED</prop>
        <prop key="setRealmForSite">PROPAGATION REQUIRED</prop>
        <prop key="*">PROPAGATION_SUPPORTS</prop>
          </props>
       </property>
   </bean>
</beans>
```

由于 Jetspeed 对 Spring 的天生依赖，很自然 Jetspeed 也支持基于 Spring MVC framework，如图 6-14 可知，Jetspeed Portal 从 J2EE 角度来看其实就是一个标准的 Web 应用程序，只不过在 Servlet 架构上引入了 Component Manager 的概念，然后用 Spring 实现了 ComponentManager 接口。因此如果不满意 Spring，更换它也是有可能的。当 Servlet 被容器停止时，也会同时关闭 SpringComponentManager。Servlet 启动完毕后，所有通过 Spring Bean XML 定义 POJO 都被实例化了，除了那些指定了 lazy init 属性为 true 的 Bean。

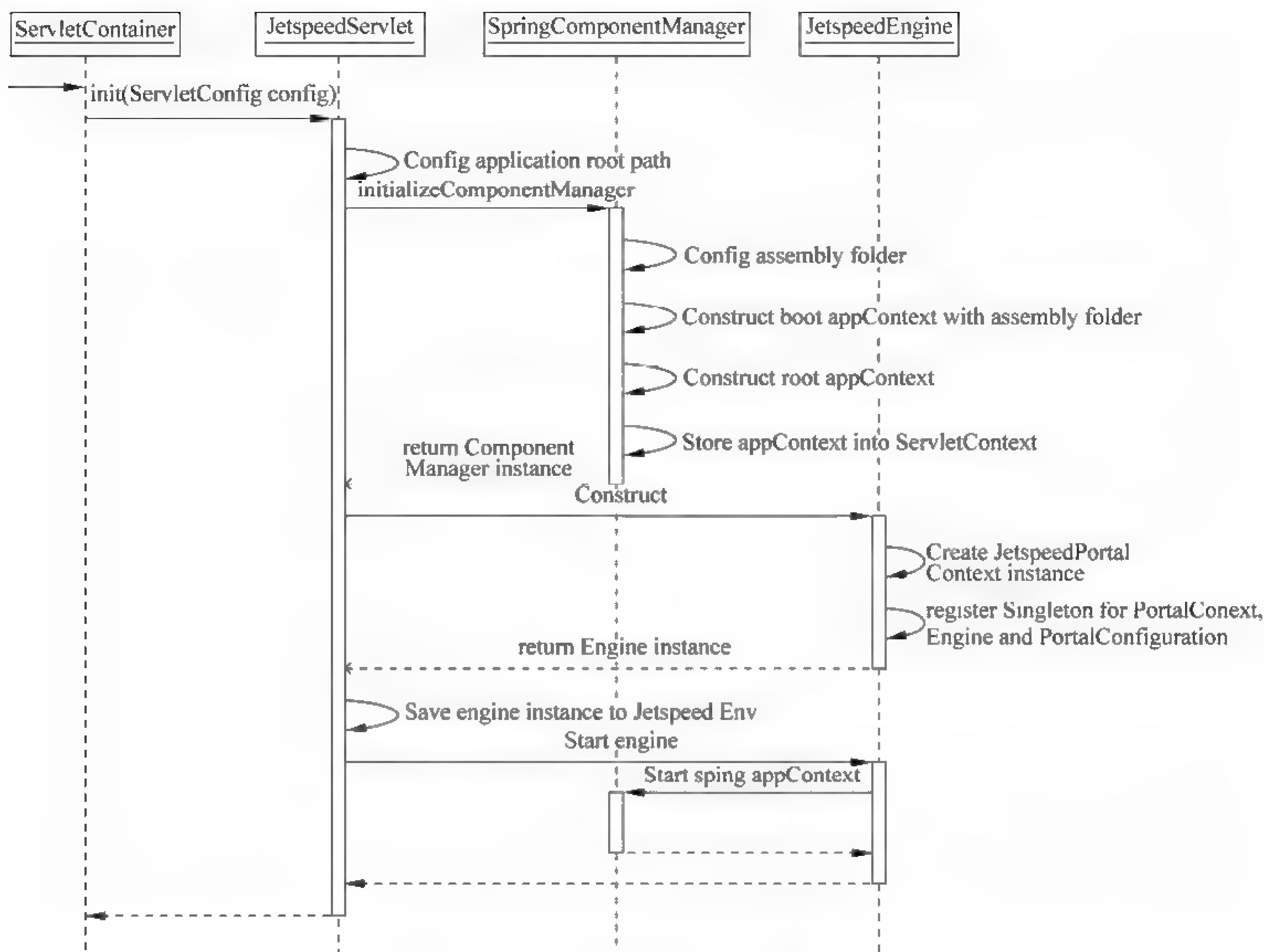

图 6-14　Jetspeed Portal 启动流程图

6.4　iweb SNS

iWebSNS[①]是一款功能强大且易于扩展的 LAMP 开源 SNS（社交网络）国产开源软件。通过 iWebSNS，可以使网站迅速、轻松地构建起一个以人际关系为核心的结构体系。从而使得用户可以通过自主创建日志、上传照片，并来记录他/她生活中的点滴故事、分享他/她的喜怒哀乐；通过好友新鲜事，用户也可以第一时间了解到众多好友的最新动态信息，增加彼此之间的交流互动。用户也可以通过加入群组来结识更多的同好伙伴，扩展自己的视野，增加自己的交际圈。作为一款大型高并发、高负载的开源 SNS 软件，它基于 iweb SuperInteraction（简称 iweb SI）框架开发。借助 iwebSI 平台，站点可以轻松获得支持热插拔及快速增加新结点的集群计算与处理能力（分布式计算与存储/高可用性/负载均衡），以方便管理 Web 2.0 类站点持续增长的数据量。同时，基于内存的分布式缓存系统、dfs（分布式文件系统）、分布式数据存储等可以轻松支持站点拥有服务于百万甚至千万级庞大用户群的能力，并且不管这些交互式服务的请求是来自计算机还是移动终端，如图 6-15 所示。

① http://www.jooyea.net/sns/index.html

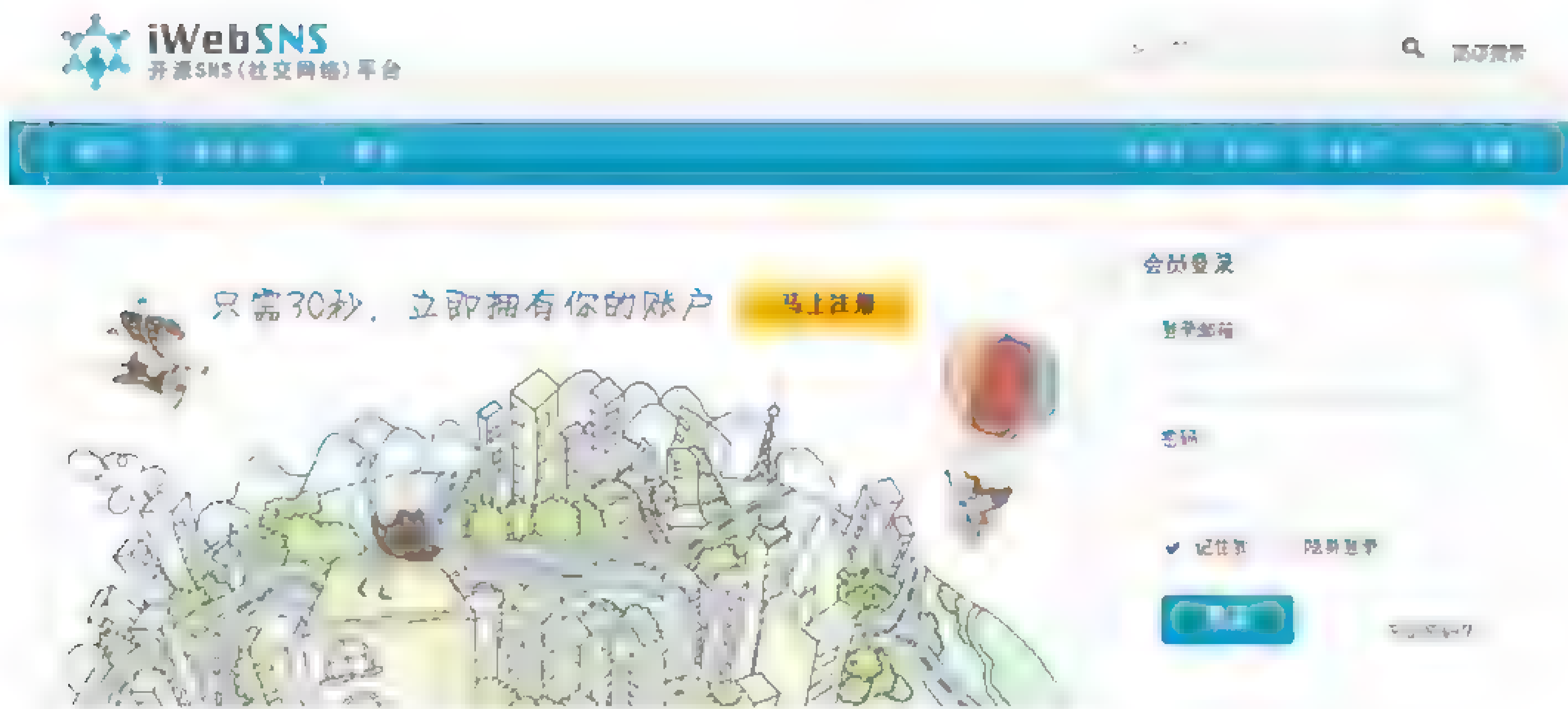

图 6-15　iweb SNS 工作界面

6.5　Struts

Struts 是 Apache 软件基金会（ASF）赞助的一个开源项目。Struts 这个名字来源于在建筑和旧式飞机中使用的支持金属架。它最初是 Jakarta 项目中的一个子项目，并在 2004 年 3 月成为 ASF 的顶级项目。它通过采用 Java Servlet/JSP 技术，实现了基于 Java EE Web 应用的 Model-View-Controller（MVC）设计模式的应用框架（Web Framework），是 MVC 经典设计模式中的一个经典产品。

在 Java EE 的 Web 应用发展的初期，除了使用 Servlet 技术以外，普遍是在 JSP 的源代码中，采用 HTML 与 Java 代码混合的方式进行开发。因为这两种方式不可避免的要把表现与业务逻辑代码混合在一起，都给前期开发与后期维护带来巨大的复杂度。为了摆脱上述的约束与局限，把业务逻辑代码从表现层中清晰的分离出来，2000 年，Craig McClanahan 采用了 MVC 的设计模式开发 Struts。后来该框架产品一度被认为是最广泛、最流行 JAVA 的 WEB 应用框架。2006 年，WebWork 与 Struts 这两个优秀的 Java EE Web 框架（Web Framework）的团体，决定合作共同开发一个新的，整合了 WebWork 与 Struts 优点，并且更加优雅、扩展性更强的框架，命名为 Struts 2，原 Struts 的 1.x 版本产品称为 Struts 1。但是很多开发人员认为 Struts 目前尚不成熟，应用的成本较高。这些缺点不能覆盖的优点，Struts 的优点主要集中体现在两个方面：Taglib 和页面导航。Taglib 是 Struts 的标记库，灵活动用，能大大提高开发效率。它的目的是为了减少在运用 MVC 设计模型来开发 Web 应用的时间。Struts 跟 Tomcat、Turbine 等诸多 Apache 项目一样，是开源软件，这是它的一大优点，使开发者能更深入的了解其内部实现机制。

6.5.1　MVC

MVC 模式是软件工程中的一种软件架构模式，把软件系统分为三个基本部分：模型

（Model）、视图（View）、控制器（Controller）。最早由 Trygve Reenskaug 在 1974 年提出，是 Xerox PARC 在 20 世纪 80 年代为程序语言 Smalltalk 发明的一种软件设计模式。它的目的是实现一种动态的程序设计，使后续对程序的修改和扩展简化，并且使程序某一部分的重复利用成为可能。除此之外，此模式通过对复杂度的简化，使程序结构更加直观。软件系统通过对自身基本部分分离的同时也赋予了各个基本部分应有的功能。下面是对 MVC 中各项做进一步描述。

（1）模型（Model）：数据模型（Model）用于封装与应用程序的业务逻辑相关的数据，以及对数据的处理方法。“模型”有对数据直接访问的权力，例如对数据库的访问。模型不依赖视图和控制器，也就是说，模型不关心它会被如何显示或是如何被操作。但是模型中数据的变化一般会通过一种刷新机制来公布。为了实现这种机制，那些用于监视此模型的视图必须事先在此模型上注册，从而，视图可以了解在数据模型上发生的改变。

（2）视图（View）：视图能够实现数据有目的的显示。在视图中一般没有程序上的逻辑。为了实现视图上的刷新功能，视图需要访问它监视的数据模型（Model），因此应该事先在被它监视的数据那里注册。

（3）控制器（Controller）：控制器起到不同层面间的组织作用，用于控制应用程序的流程。它处理事件并作出响应，事件包括用户的行为和数据模型上的改变。

对 MVC 的划分从最初基于 JSP 的 Model1、到基于 MVC 的 Model2 框架，以及满足当前 AJAX 和门户技术的 Model3。

6.5.1.1 Model1

Model1 模型是以 JSP 为中心开发的，即 Web 应用由一组 JSP 页面构成，在页面里嵌入<jsp:useBean>、<jsp:script>和<html>等，实现页面显示、业务逻辑和流程控制。但出现可操作性差、扩展性弱和更新难，因为一组 JSP 页面实现一个业务流程；业务逻辑和表现逻辑没有进行抽象和分离，耦合度高，因此可重用性差，移植性弱，不利于改动，导致这种原因是因为没有采用模块的思想；这种模型（Model1）一般适合于小型 Web 应用开发。鉴于这些不足，出现了基于 MVC 的 Model2 模型；其中 M 表示用 JavaBean、EJB 组件等实现业务逻辑，V 表示用 JSP 产生页面，C 表示用 Servlet 实现过程控制。这样把应用逻辑、处理过程和显示逻辑用不同的组件实现，就解决了重用和交互以及不同程度改善了耦合度。

6.5.1.2 Model2

Struts 是 MVC 的 Model2 一种实现，主要采用技术是 Servlet、JSP 和自定义标记库，并且 Struts 对 MVC 提供对应实现组件，因此降低了表现逻辑与业务逻辑以及业务逻辑与数据接口之间的耦合。其中 M 表示应用程序状态，主要通过 ActionServlet 创建并使用 Action 和 ActionForm bean 对象，其配置可以在 Web.xml 中进行；Action 直接处理数据交互，而 ActionForm bean 组件实现对 V 与 M 之间交互支持，即 JSP 文件用 JSP 标记来读取 ActionForm bean 信息；通常用 JavaBean 表示系统内部状态，同时根据系统的复杂度也可以使用 Entity EJB 和 Session EJB 等组件实现与数据接口持久性和维持应用的状态。V 表示通过 JSP 标记完成应

用逻辑与表现逻辑分离，以及实现与 M 中的 ActionForm bean 的映射和完成对用户数据的封装等。C 作用是从客户端接受请求，并且选择执行相应的业务逻辑，然后把响应结果送回到客户端，主要由 ActionServlet 和 ActionMapping 对象组成，使用 Servlet 类型的 ActionServlet 接受客户端请求，且 ActionServlet 包括基于配置文件（struts-config.xml）的 ActionMapping 对象，每个对象实现一个请求到具体的 M 中的 Action 之间映射，如图 6-16 所示。

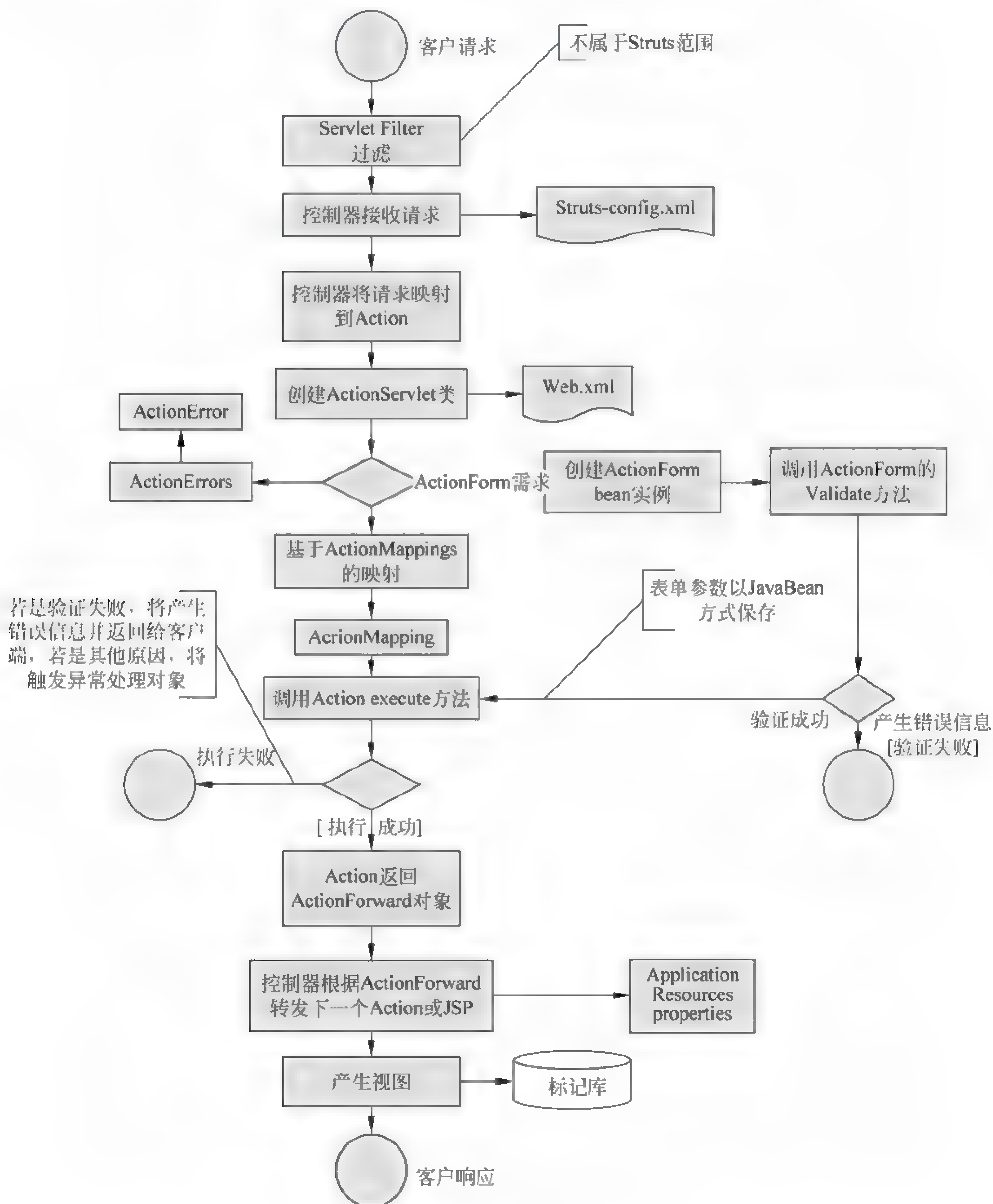

图 6-16　Struts 执行流程图

6.5.1.3 Model3

StrutsCX[①]是基于Struts的一个Web开源框架，它降低了Struts中JSP使用Java代码，包括大量标记和Struts应用框架等；是使用XML、XSL、XSLT和Xpath等技术和Struts框架开发Web应用，同时可以不再使用JSP，即StrutsCX是使用XSLT而不是JSP作为表示层的Struts，其转换原理是通过StrutsCXDocumentBuilder和StrutsCXTransformer来实现，因此可以将Struts中的FormBean、Session、ArrayList等需要显示的内容映射成XML内容，在XSL中通过XSLT、Xpath等技术访问，从而使表现层中的HTML与Java代码分离。而且StrutsCX通过配置后可以输出XML、PDF、Ascil等多格式，所以扩展性好。因此StrutsCX具有这些特点：

（1）任何Servlet容器中都可以运行轻量级Web开发框架；

（2）可以完全使用XML、XSLT、Xpath等代替JSP和文本信息保存，同时完全支持XML、XHTML、XSL-FO、WML等格式输出，以及内置的FOP支持PDF、SVG、ASCLL等格式生成；

（3）有简单校验机制，包含Struts校验，并且默认使用JDOM；

（4）所有访问都通过映射为*.do来实现。如图6-17所示。

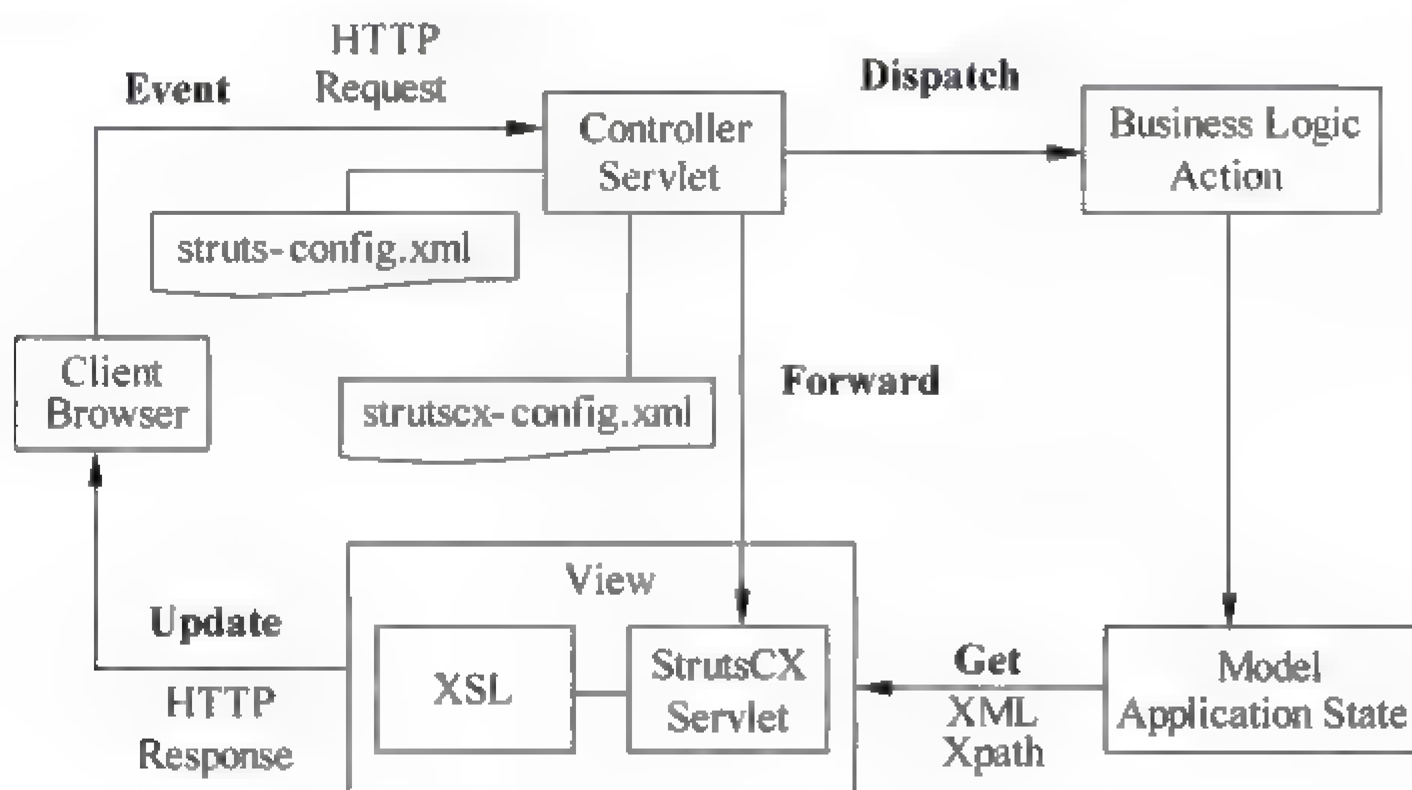

图6-17 StrutsCX概览

而MVC的支持下通过把Ajax和门户技术引入到StrutsCX框架中去，称为Model3。这样将有效解决Model2的局限性，其主要在M、V和通信方式中进行改进，即在M中引入DWR完成Bean等组件向JavaScript转换，实现门户容器定制，并实现页面实时性；在V中引入CSS实现界面系统动态变化显示；通信方式采用XMLHttpRequest实现异步操作。因此，表现逻辑与业务逻辑的耦合度大大的降低，分离程度大大提高，业务流程越来越松散，因此一般与Web服务结合后，适合构建大型集成系统，如图6-18、图6-19所示。

① http://it.cappucc-inonet.com/strutscx/index.php

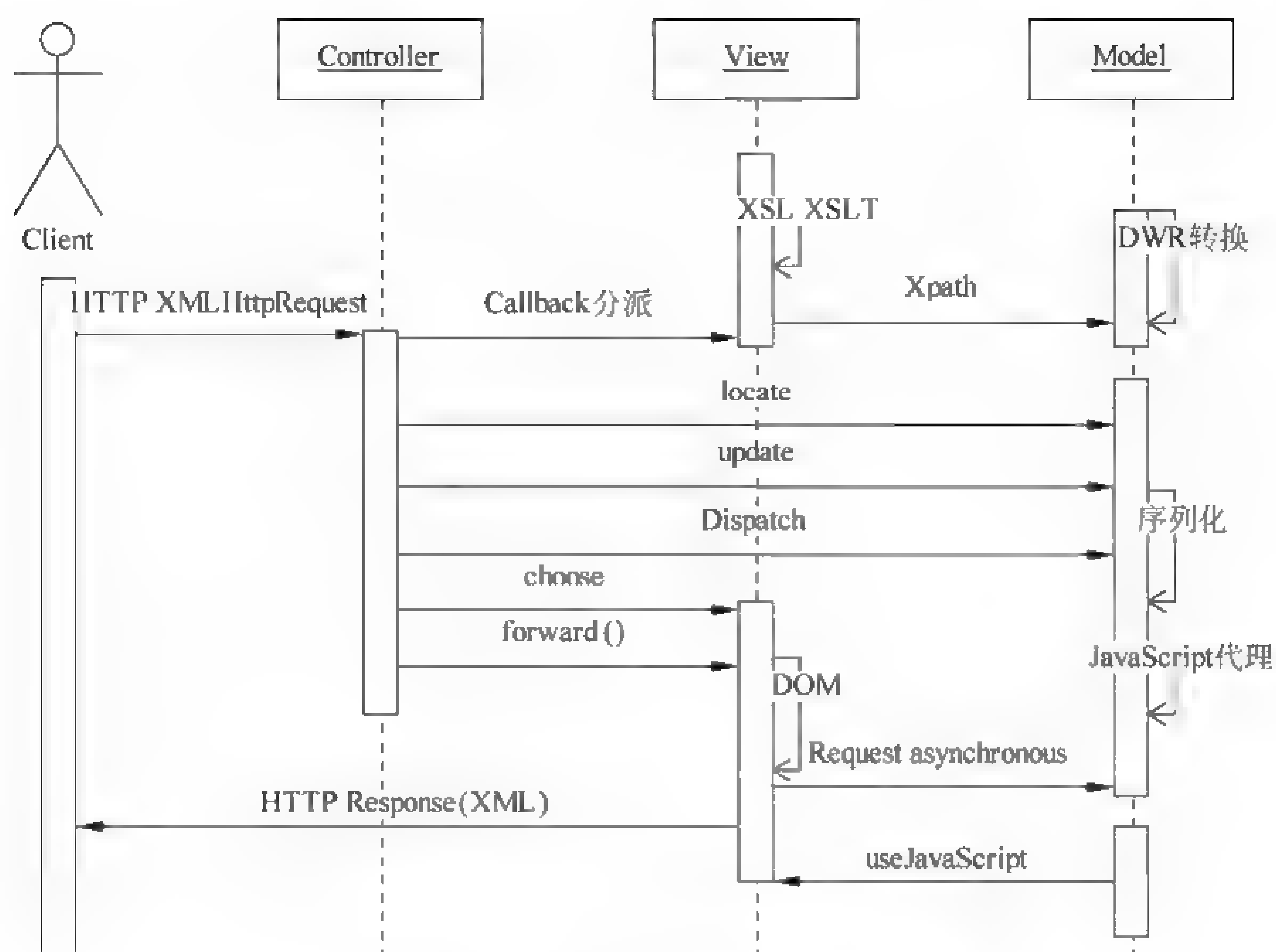

图 6-18　Model3 运行模式概图

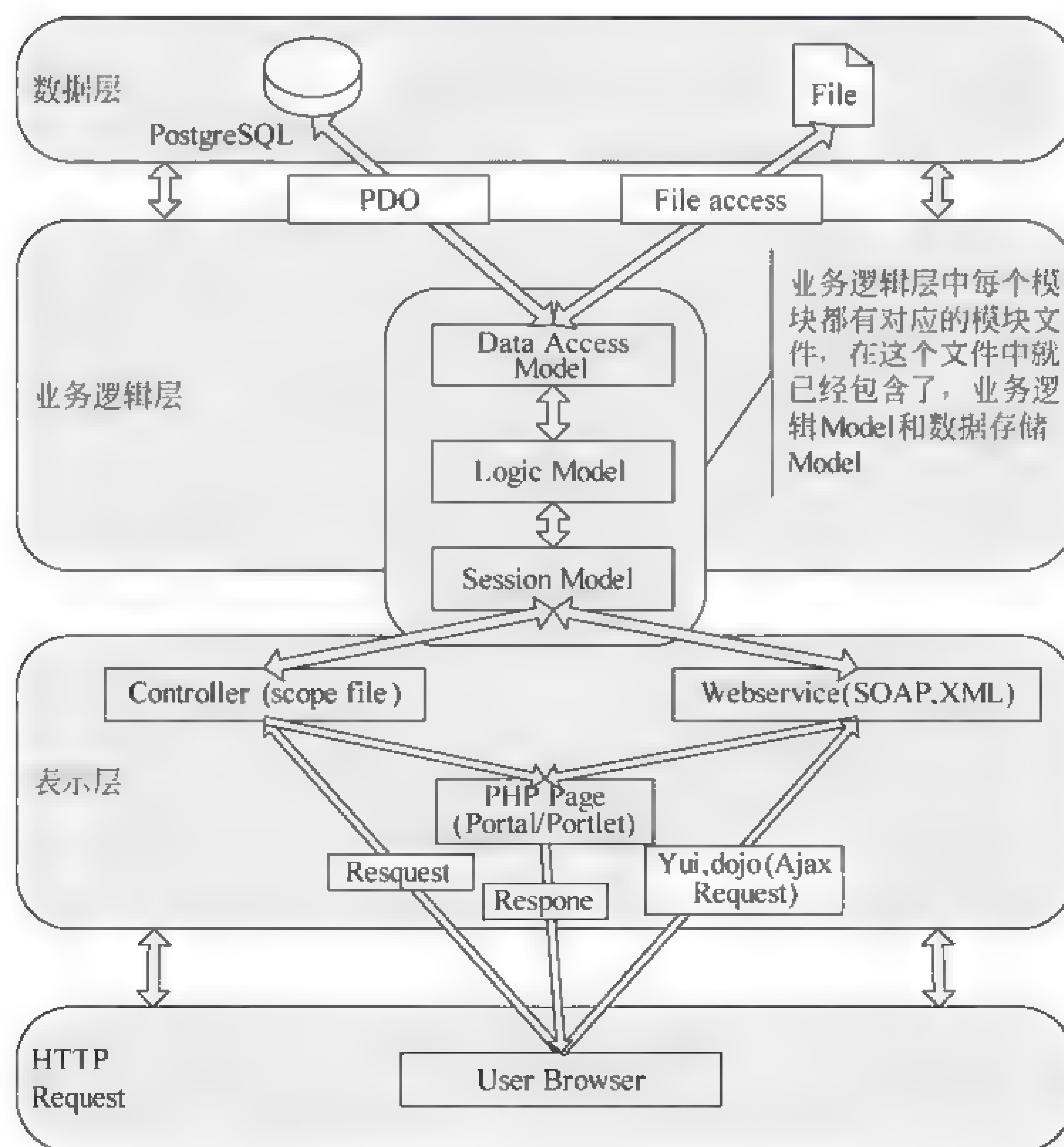

图 6-19　基于 Model3 的 MVC 构架图

6.5.2 Struts 应用方法

Struts 是一种开源框架，可用来构建 Web 应用程序，它基于流行的 Model-View-Controller (MVC2)设计范型。该框架构建在一些标准的技术之上，比如 Java Servlets、JavaBeans、ResourceBundles 和 XML，并且可提供灵活和可扩展的组件。Struts 以 ActionServlet 的形式实现了 Controller 层，并建议使用 JSP 标记库构建 View 层。Struts 通过 Action 类提供了围绕 Model 层的包装器。图 6-20 所示是基于 MVC 的 Struts 结构，图 6-21 所示是 Struts 工作流程。

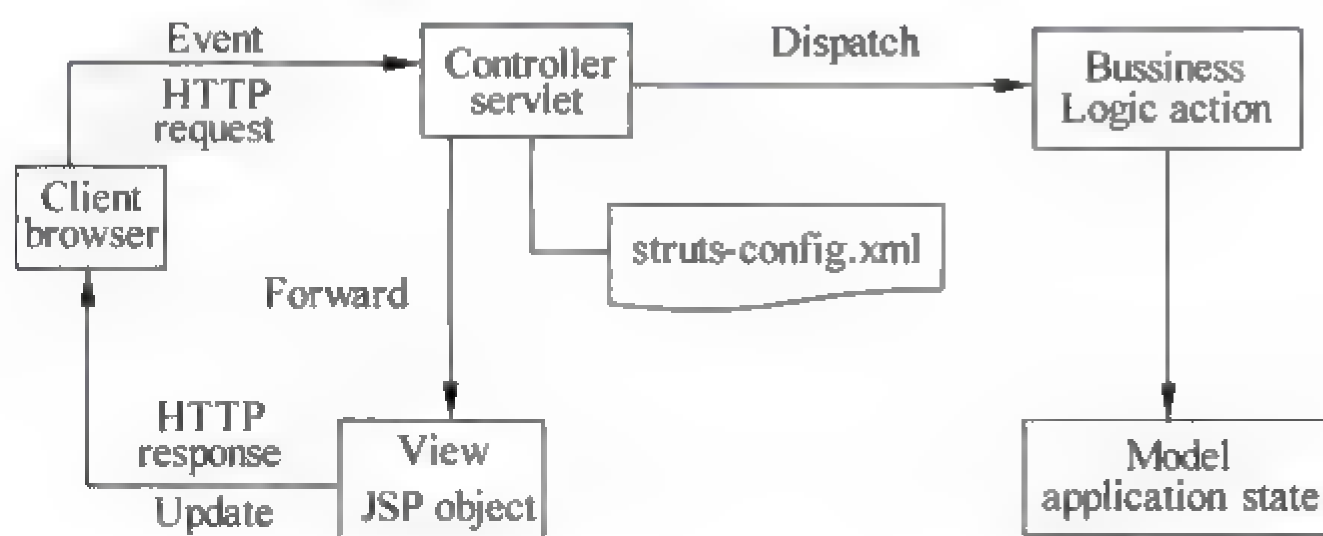

图 6-20　基于 MVC 的 Struts 结构

6.5.2.1 Struts 应用基础

在 Struts 常用的类之前，先介绍两个重要的配置文件：web.xml 和 struts-config.xml，通过它们配置 Struts 系统中的各个模块之间的交互，程序片段（具体详细配置参见后续章节）如下：

```
<servlet>
    <servlet-name>action</servlet-name>
<servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    ...
</servlet>
```

表示在 web.xml 中述了系统的 Controller 对象。

```
<servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
</servelt-mapping>
```

表示在 web.xml 中的 servlet 对象就是 Struts 提供的 Controller。

```
<taglib>
    <taglib-url>/WEB-INF/struts-bean.tld</taglib-url>
    <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
</taglib>
```

表示在 web.xml 中实现客户请求的 url 信息、标记库和服务器端具体处理的映射关系。

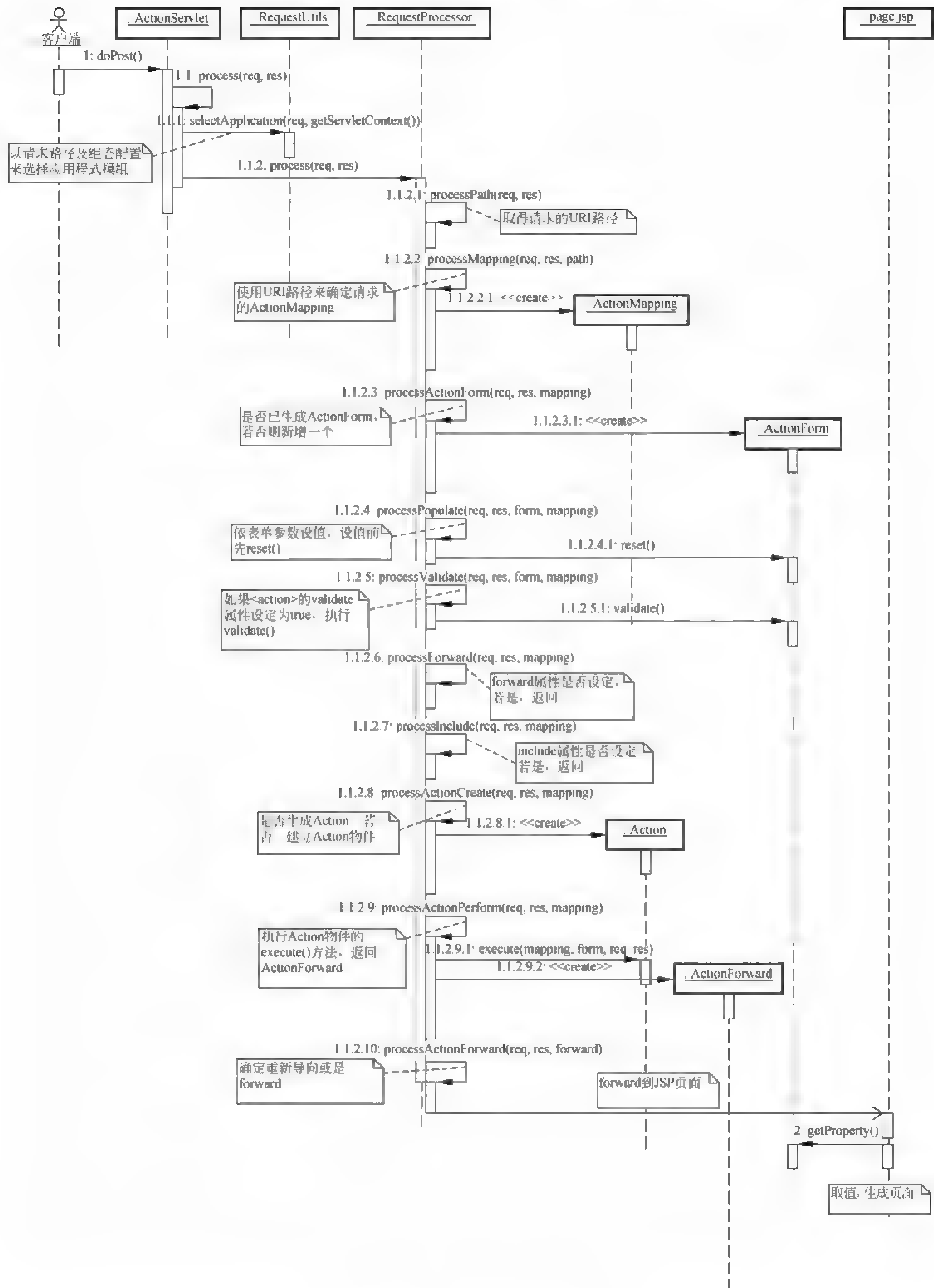

图 6-21　Struts 工作流程

struts-config.xml 是用于建立 Controller 和 Model 之间的关系的。它描述了 Controller 所使用的把请求对应到具体处理的法则，同时它还描述了客户提供的数据与 ActionForm 组件的对应映射关系。

```
<form-beans>
    <form-bean name="loginForm" type="loginForm" />
</form-beans>
```

在 struts-config.xml 中的<form-bean>表示标记描述一个具体的 ActionForm 子类对象，通过它和 JSP 页面中的自定标记的结合使用可以实现 ActionForm 和 View 之间的数据映射。

```
<action-mappings>
    <action path="/login" type="loginAction" name="loginForm" input=
    "/login.jsp" />
</action-mappings>
```

在 struts-config.xml 中的<action-mappings>表示标记描述了请求和处理的一对一映射关系。input 和 path 属性唯一的标记了客户端的一个请求，name 属性描述封装客户端数据的 ActionForm 子类对象，Type 属性描述处理这个请求的 Action 子类对象。

下面介绍 Struts 常用类的基本情况：

1．ActionServlet 类

ActionServlet 是该 MVC 实现的 Command 部分，它是这一框架的核心。ActionServlet（Command）创建并使用 Action、ActionForm 和 ActionForward。在 struts-config.xml 文件中配置该 Command。在创建 Web 项目时，将扩展 Action 和 ActionForm 来解决特定的问题。其 struts-config.xml 的目的是指示 ActionServlet 如何使用这些扩展的类。

同时，应用程序的每个 Action 都会扩展 Struts 的 org.apache.struts.action.Action 类。这些 Action 类为应用程序的 Model 层提供了一个接口，充当围绕业务逻辑的包装器。每个 Action 类都必须向 perform() 方法提供其特定于用例的实现。perform() 方法经常返回类型是 ActionForward 的一个值。

而应用程序的 ActionForm 扩展了 Struts 的 org.apache.struts.action.ActionForm 类。ActionForm 是一些封装和验证请求参数的简单 JavaBean。当要验证请求数据时，ActionForm 的 validate()方法必须给出一个特定于该情况的实现。ActionForm 作为运载工具，向 Action 类提供请求数据。一个 JSP 对象与各自的 ActionForm 对象相结合，构成应用程序的 View 层。在该层，几乎 JSP 对象的每个表单字段都映射到相应的 ActionForm 的属性。

Struts 建议将每个 JSP 对象与一个 ActionForm 相关联，后者可以封装屏幕上显示的数据。并通过 ActionForm 内的附加方法来访问 JSP 对象内的表单数据。如下程序片段 ActionForm 标记在 View 层中的方法：

```
<html:form action="/bp1">
  <bean:define name="bp1AForm" property="bp1BForm" id="bp1B"
            type="com.test.dw.webarch.struts.BP1BForm" />
  <html:text  name="bp1B" property="subsetAtt1" />
</html:form >
```

在此程序片段中的这个 ActionForm 被称为 BP1AForm，它包括属性 attrib1 及其 getter 和 setter 方法。在配置文件 struts-config.xml 中，行为/bp1 通过 name 属性映射到 bp1AForm。这有助于在 JSP 中显示数据。创建一个 JavaBean（BP1BForm），其属性是 BP1AForm 属性的子集，并且通过将这个 bean 与 BP1AForm 关联，用 bean BP1BForm 的属性替代 BP1AForm 中的属性。现在就可以通过 BP1BForm 访问 BP1AForm 中的属性子集了。

这种实践的主要优势是可用于多个 ActionForm 访问一个属性集。这是因为 Struts 实现了 <bean:define/> 标记。当代码 <%@ taglib uri="struts-bean.tld" prefix="bean"%> 指向 struts-bean.tld 时，<bean:define/> 标记开始在 JSP 组件内工作。并由 ActionForm 扩展而来的 BP1AForm 验证框架必须验证 BP1BForm 的数据。

同时，当在应用程序中创建 Action 类时，不需要直接扩展 org.apache.struts.action.Action，可以通过扩展 org.apache.struts.action.Action 创建一个 Action 类（IntermediateAction），用于处理应用程序中的常见事务。当然所有其他的 Action 类都扩展了 IntermediateAction 类。

2．ActionForm 类

ActionForm 维护 Web 应用程序的会话状态。ActionForm 是一个抽象类，必须为每个输入表单模型创建该类的子类。即输入表单模型时，是指 ActionForm 表示的是由 HTML 表单设置或更新的一般意义上的数据。并且需要在配置 struts-config.xml 文件中控制 HTML 表单请求与 ActionForm 之间的映射关系，将多个请求映射到 UserActionForm，实现 UserActionForm 可跨多页进行映射，以执行诸如向导之类的操作。在 ActionForm 实现会话处理时，大多数 Web 应用程序都在会话中保持数据，使其在整个应用程序过程中可用。这种最佳实践实现了这种 Web 应用程序特性。它允许方法 toSession()和 fromSession()将会话数据移动到表单数据或从表单数据移回。因此，它实现了在 Web 应用程序中保持会话数据。Struts 框架将执行以下操作，如图 6-22 所示。

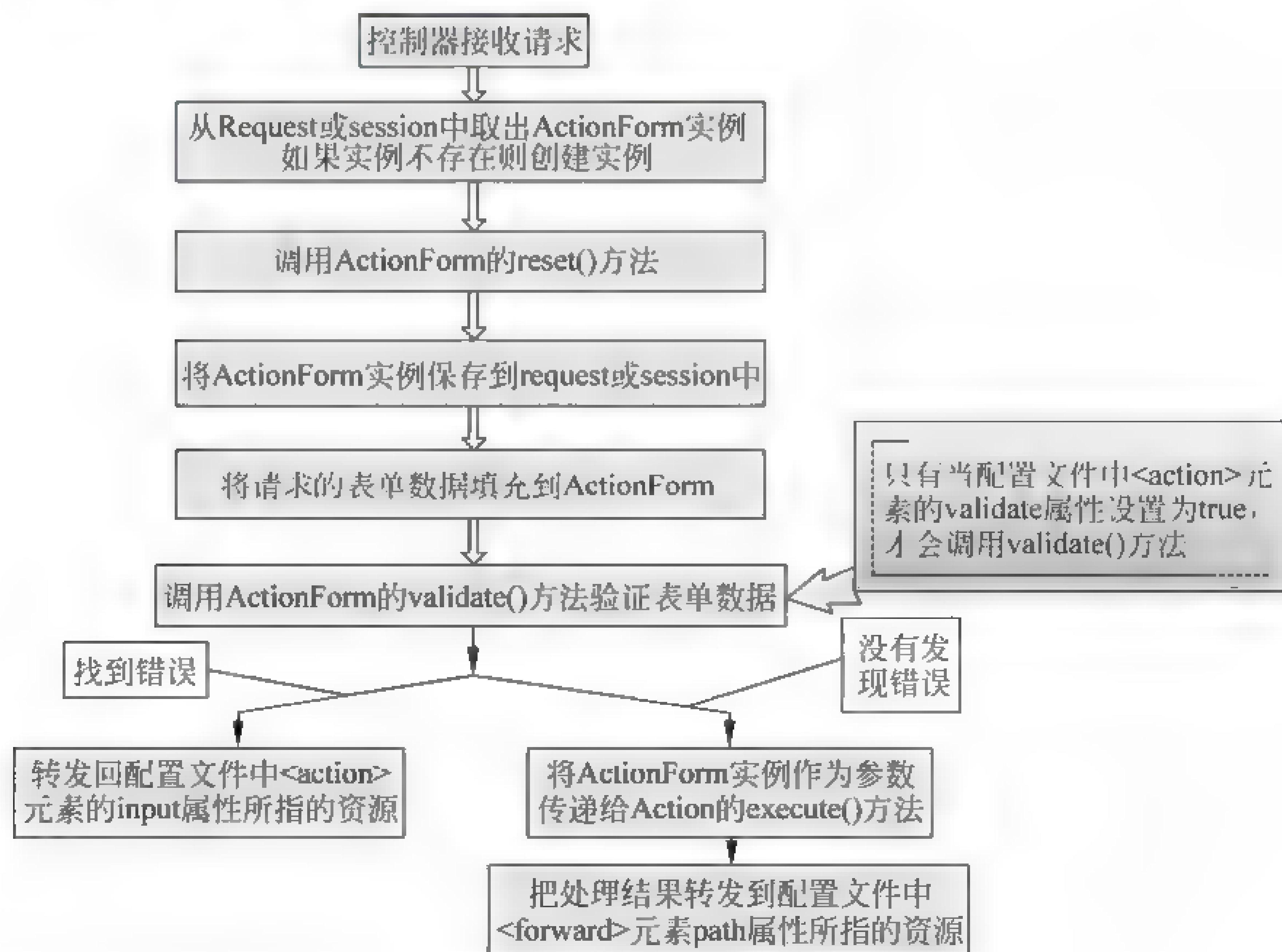

图 6-22　ActionForm 搜索过程

在图 6-22 中：

（1）检查 UserActionForm 是否存在；如果不存在，它将创建该类的一个实例。

（2）Struts 将使用 HttpServletRequest 中相应的域设置为 UserActionForm 的状态。例如，Struts 框架将从请求流中提取 fname，并调用 UserActionForm.setFname()。

（3）Struts 框架将 UserActionForm 传递给业务包装 UserAction 之前，进行更新它的状态。

（4）将它传递给 Action 类之前，Struts 还会对 UserActionForm 调用 validation()方法进行表单状态验证。别的网页或业务可能使用 UserActionForm，在这些地方，验证可能有所不同。在 UserAction 类中进行状态验证可能更好。

（5）可在会话级维护 UserActionForm。

3．Action 类

Action 类是业务逻辑的一个包装。Action 类的用途是将 HttpServletRequest 转换为业务逻辑。要使用 Action，需要创建它的子类并覆盖 process()方法。Action 类应该控制应用程序的流程，而不应该控制应用程序的逻辑。

考虑 Action 类的另一种方式是 Adapter 设计模式。Action 的用途是将类的接口转换为客户机所需的另一个接口。Adapter 使类能够协同工作，如果没有 Adapter，则这些类会因为不兼容的接口而无法协同工作。Struts 提供了它能够理解的一个业务接口，即 Action。通过扩展 Action，使得业务接口与 Struts 业务接口保持兼容。

但通常在使用这个 Struts 框架时，对于 JSP 组件请求应用程序执行的每个动作，应用程序都必须扩展 Struts 的 org.apache.struts.action.Action 以创建 Action 类。在处理请求时，单个的 Action 类与应用程序的 Model 层连接，操作流程如下：

（1）通过扩展 org.apache.struts.action.Action 创建一个 Action 类，比如 BP2Action。

（2）通过扩展 BP2Action 在 Web 应用程序中创建所有其他 Action 类。

（3）在 BP2Action 类中创建一个方法 performTask()，就像在公共抽象类 ActionForward performTask(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException 中一样。

（4）在 BP2Action 中向应用程序添加一个或多个泛型方法，比如 serverSideValidate()。

考虑以下（5）～（8）因素后决定方法的访问修饰符：

（5）如果所有 Action 类都必须实现此方法，则让其为抽象。

（6）如果某些 Action 类提供一个特定的实现，则将此方法声明为受保护，并给它一个默认实现。

（7）在 BP2Action 类中，将方法 perform()声明为 final。调用上述的泛型方法（通常在处理请求前调用该方法）。现在就可以调用第（3）项中创建的方法 performTask()。

（8）在每个扩展 BP2Action 的 Action 类，添加具有特定实现的方法 performTask()。

4．ActionErrors 类

可以使用 ctionError 来支持异常处理。ActionError 用来捕捉应用程序异常，并将其传送给 View 层。每个异常都是一个 ActionError 实例的集合。ActionError 可以封装错误消息，而 Presentation 层中的</html:errors>可以呈现 ActionError 集合内的所有错误消息。ActionErrors 是 ActionError 类的容器，View 可以使用标记访问这些类。ActionError 是 Struts 保持错误列表的方式。在传统的 Action 类中发生应用程序异常时，异常首先被写入日志。然后此类创建

一个 ActionError，并在合适的作用域中存储它。然后 Action 类再将控制转交给合适的 ActionForward。如下程序片段展示了 Action 类是如何处理异常的：

```
try {
   //Code in Action class
   }
   catch (ApplicationException e) {
   //log exception
   ActionErrors actionErrors = new ActionErrors();
   ActionError actionError = new ActionError(e.getErrorCode());
   actionErrors.add(ActionErrors.GLOBAL_ERROR, actionError);
   saveErrors(request, actionErrors);
   }
```

一般采用如下步骤来实现应用：

（1）通过扩展 org.apache.struts.action.Action 创建一个 Action 类，比如 BP4Action。

（2）通过扩展 BP4Action 在 Web 应用程序中创建所有其他 Action 类。

（3）在 BP4Action 中声明变量 ActionErrors actionErrors = new ActionErrors();。

（4）在 BP4Action 中创建方法 performTask()，就像在公共抽象类 ActionForward performTask(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response, ActionErrors actionErrors) throws IOException, ServletException 中一样。

（5）在 BP4Action 中将方法 perform()声明为 final。然后调用泛型方法（这些方法总是在处理请求前调用）。现在调用在前一个步骤中的 performTask()，并创建它。

（6）在每个 Action 类中实现方法 performTask()（通过扩展 BP4Action），如下程序片段那样处理应用程序异常：

```
try{
   //Code in Action class
   }
   catch(ApplicationException appException) {
   //Log exception
   //Add error to actionErrors
   actionErrors.add(ActionErrors.GLOBAL ERROR,
     new ActionError(appException.getErrorCode()));
   }
```

在程序片段的 BP4Action 中，调用方法 performTask()之后，通过 saveErrors(request, errors)保存 ActionErrors。

5. ActionMapping 类

输入事件通常是在 HTTP 请求表单中发生的，servlet 容器将 HTTP 请求转换为 HttpServletRequest。控制器查看输入事件并将请求分派给某个 Action 类。struts-config.xml 确定 Controller 调用哪个 Action 类。struts-config.xml 配置信息被转换为一组 ActionMapping，而后者又被放入 ActionMappings 容器中。ActionMapping 包含有关特定事件如何映射到特定

Action 的信息。ActionServlet（Command）通过 perform() 方法将 ActionMapping 传递给 Action 类。这样就使 Action 可访问用于控制流程的信息。而 ActionMappings 是 ActionMapping 对象的一个集合。

6. JSP 定制标记库

JSP 定制标记库是用标记表示的一组行为的集合。这是 JSP Specification 1.1 的一个强大特性；它将其他应用程序层与表示层分离出来。这些库易于使用，而且可以以一种类似 XML 的方式来读取。只要尽量少地在其中使用 Javascriptlet，就可以轻松维护 JSP 组件。Struts 提供的 JSP 标记包括 HTML、逻辑和 bean 标记。而创建一个标记并将放入到标记库，可以按经下步骤来完成：

（1）创建标记，将 scriptlet 代码从原先所在的地方移到一个 Java 类（LastModifiedTag）中，程序片段如下：

```
package com.newInstance.site.tags;
import java.io.File;
import java.io.IOException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.jsp.tagext.TagSupport;
public class LastModifiedTag extends TagSupport { }
```

（2）创建一个标记库描述符（tag library descriptor，TLD）文件。TLD 向容器和任何要使用该标记库的 JSP 页面描述标记库。如下程序片段是一个非常标准的 TLD 例子，其中只包含了一个标记。

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE taglib
    PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2/EN"
           "http://java.sun.com/dtd/web-jsptaglibrary 1 2.dtd">
<taglib>
    <tlib-version>1.0</tlib-version>
    <jsp-version>1.2</jsp-version>
    <short-name>site-utils</short-name>
    <uri>http://www.newInstance.com/taglibs/site-utils</uri>
    <tag>
      <name>lastModified</name>
      <tag-class>com.newInstance.site.tags.LastModifiedTag</tag-class>
      <body-content>empty</body-content>
    </tag>
</taglib>
```

（3）保存这个文件，并将其放到 WEB-INF/tlds 目录下，将这个文件保存为 site-utils.tld，并在该标记库的 URI（推荐的前缀）和 TLD 文件本身之间再次创建一个链接。对于这个特定的标记库，要使其可以使用，最后一步要做的是让 Web 应用知道如何连接一个 JSP 页面中的 URI，如何请求使用一个标记库。这可以通过应用的 web.xml 文件来做。如下程序片段显示了一个非常简单的 web.xml：

```
<taglib>
    <taglib-uri>http://www.newInstance.com/taglibs/site utils</taglib-uri>
    <taglib-location>/WEB-INF/tlds/site-utils.tld</taglib-location>
</taglib>
```

（4）现在就可在 JSP 页面中引用新标记了。如下程序片段展示了新改进的 footer.jsp，这个文件中现在完全没有 scriptlet，也没有指向具有 scriptlet 的 JSP 页面的引用：

```
<%@ taglib prefix="site-utils"
           uri="http://www.newInstance.com/taglibs/site-utils" %>
          </td>
          <td width="16" align="left" valign="top"> </td>
    </tr>
        <tr>
      <td width="91" align="left" valign="top" bgcolor="#330066"> </td>
      <td align="left" valign="top"> </td>
      <td class="footer" align="left" valign="top"><div align="center"><br>
        &copy; 2003
        <a href="mailto:webmaster@newInstance.com">Brett McLaughlin</a><br>
        Last Updated: <site-utils:lastModified />
        </div></td>
      <td align="left" valign="top"> </td>
      <td width="141" align="right" valign="top" bgcolor="#330066"> </td>
    </tr>
</table>
```

6.5.2.2　Struts 在 RESTful Web Services 中应用方法

在前面的章节已经描述了 RESTful Web Services 的原理和基本应用方法，下面继续描述在 Struts 2 中开发 RESTful Web Services 的方法。而 Struts 2 开始提供 Convention 插件，它允许根据约定来搜索 Action，以及管理 Action 和 Result 的映射。另外，Struts 2.1 还提供了 REST 插件，使 Struts 2 可以支持 Rails 风格的 URL，以对外提供 REST 风格的资源服务。Convention 插件彻底地抛弃了配置信息，不仅不需要使用 struts.xml 文件进行配置，甚至不需要使用 Annotation 进行配置。而是 Struts 2 根据约定来自动配置。而 REST 插件的核心是 RestActionMapper，它负责将 Rails 风格的 URL 转换为传统请求的 URL，并且 Struts 2 的 REST 插件默认支持 XHTML、XML 和 JSON 三种形式的数据。这时在 struts2-rest-plugin 中找到 struts-plugin.xml 进行配置 RestActionMapper，程序片段如下：

```
<bean type="org.apache.struts2.dispatcher.mapper.ActionMapper"
  name="rest" class="org.apache.struts2.rest.RestActionMapper" />
```

可以使用如下步骤完成 Struts 2 对 RESTful Web Services 应用安装：

（1）将 Struts 2 项目下 struts2-convention-plugin-2.1.6.jar、struts2-rest-plugin-2.1.6.jar 两个 JAR 包复制到 Web 应用的 WEB-INF\lib 路径下。

（2）由于 Struts 2 的 REST 插件还需要提供 XML、JSON 格式的数据，因此还需要将 xstream-1.2.2.jar、json-lib-2.1.jar、ezmorph-1.0.3.jar，以及将 Jakarta-Common 相关 JAR 包复制到 Web 应用的 WEB-INF/lib 路径下。

（3）通过 struts.xml、struts.properties 或 web.xml 改变 struts.convention.default. parent. package 常量的值，让支持 REST 风格的 Action 所在的包默认继承 rest-default，而不是继承默认的 convention-default 父包。

在 RestActionMapper 命名空间中可看到 setIdParameterName、setIndexMethodName、setGetMethodName 、 setPostMethodName 、 setPutMethodName 、 setDeleteMethodName 、setEditMethodName、setNewMethodName 等方法，这些方法对应为下面列出的方法提供 setter 支持，其中对 ID 的识别主要有以下几个方法：index（处理不带 id 请求参数的 GET 请求），show（处理带 id 请求参数的 GET 请求），create（处理不带 id 请求参数的 POST 请求），update（处理带 id 请求参数的 PUT 请求），destroy（处理带 id 请求参数的 DELETE 请求），edit（处理带 id 请求参数，且指定操作 edit 资源的 GET 请求），editNew（处理不带 id 请求参数，且指定操作 edit 资源的 GET 请求）。分别详细说明如下：

（1）struts.mapper.idParameterName：用于设置 ID 请求参数的参数名，该属性值默认是 id。

（2）struts.mapper.indexMethodName：设置不带 id 请求参数的 GET 请求调用 Action 的哪个方法，该属性值默认是 index。

（3）struts.mapper.getMethodName：设置带 id 请求参数的 GET 请求调用 Action 的哪个方法，该属性值默认是 show。

（4）struts.mapper.postMethodName：设置不带 id 请求参数的 POST 请求调用 Action 的哪个方法，该属性值默认是 create。

（5）struts.mapper.putMethodName：设置带 id 请求参数的 PUT 请求调用 Action 的哪个方法，该属性值默认是 update。

（6）struts.mapper.deleteMethodName：设置带 id 请求参数的 DELETE 请求调用 Action 的哪个方法，该属性值默认是 destroy。

（7）struts.mapper.editMethodName：设置带 id 请求参数、且指定操作 edit 资源的 GET 请求调用 Action 的哪个方法，该属性值默认是 edit。

（8）struts.mapper.newMethodName：设置不带 id 请求参数、且指定操作 edit 资源的 GET 请求调用 Action 的哪个方法，该属性值默认是 editNew。

如下程序片段配置 struts.xml 是 REST 的 Action 类使用方法：

```
<constant name="struts.convention.action.suffix" value="Controller"/>
 <?xml version="1.0" encoding="GBK" ?>
 <!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.1//EN"
    "http://struts.apache.org/dtds/struts-2.1.dtd">
 <struts>
    <constant name="struts.i18n.encoding" value="GBK"/>
    <constant name "struts.convention.action.suffix" value "Controller"/>
    <constant name "struts.convention.action.mapAllMatches" value "true"/>
    <constant name "struts.convention.default.parent.package" value "rest-
```

```
    default"/>
</struts>
```

6.6　Spring

Spring Framework 是一个开源的 Java/Java EE 全功能栈（full-stack）的应用程序框架，以 Apache 许可证形式发布，也有.NET 平台上的移植版本。该框架基于 Expert One-on-One Java EE Design and Development（ISBN 0-7645-4385-7）一书中的代码，最初由 Rod Johnson 和 Juergen Hoeller 等开发。Spring Framework 提供了一个简易的开发方式，这种开发方式，将避免那些可能致使底层代码变得繁杂混乱的、大量的属性文件和帮助类。Spring 具有能够让这部分工作变得简单的能力。程序开发员们可以使用 Spring 的 JDBC 抽象层重新设计那些复杂的框架结构。Spring 具有以下特性：

（1）这是基于 JavaBeans 的，且采用控制翻转（Inversion of Control，IoC）原则的配置管理，使得应用程序的组建更加快捷简易。

（2）一个可用于从 applet 到 Java EE 等不同运行环境的核心 Bean 工厂。

（3）数据库事务可以是一般化的抽象层，允许声明式（Declarative）事务管理器，简化事务的划分，从而使之与底层无关。

（4）针对 JTA 和单个 JDBC 数据源提供了一般化策略，使 Spring 的事务支持不要求 Java EE 环境，这与一般的 JTA 或者 EJB CMT 相反。

（5）JDBC 抽象层提供了有针对性的异常等级（不再从 SQL 异常中提取原始代码），简化了错误处理，大大减少了程序员的编码量。再次利用 JDBC 时，程序员无须再写出另一个 finally 模块。并且面向 JDBC 的异常与 Spring 通用数据访问对象 DAO（Data Access Object）异常等级相一致。

（6）以资源容器、DAO 实现和事务策略等形式与 Hibernate，JDO 和 iBATIS SQL Maps 集成。利用众多的翻转控制方便特性来全面支持，解决了许多典型的 Hibernate 集成问题。所有这些全部遵从 Spring 通用事务处理和通用数据访问对象异常等级规范。

（7）基于核心 Spring 功能的 MVC 网页应用程序框架。开发者通过策略接口将拥有对该框架的高度控制，因而该框架将适应于多种呈现（View）技术，例如 JSP，FreeMarker，Velocity，Tiles，iText 和 POI。同时，Spring 中间层可以轻易地结合于任何基于 MVC 框架的网页层，例如 Struts，WebWork 或 Tapestry。

（8）提供诸如事务管理等服务的面向方面编程框架。

6.6.1　Spring 框架介绍

Spring 是一个开源框架，是为了解决企业应用程序开发复杂性而创建的。框架的主要优势之一就是其分层架构，分层架构允许选择使用哪一个组件，同时为 J2EE 应用程序开发提供集成的框架。它由七个定义良好的模块组成。Spring 模块构建在核心容器之上，核心容器定义了创建、配置和管理 bean 的方式，如图 6-23 所示。

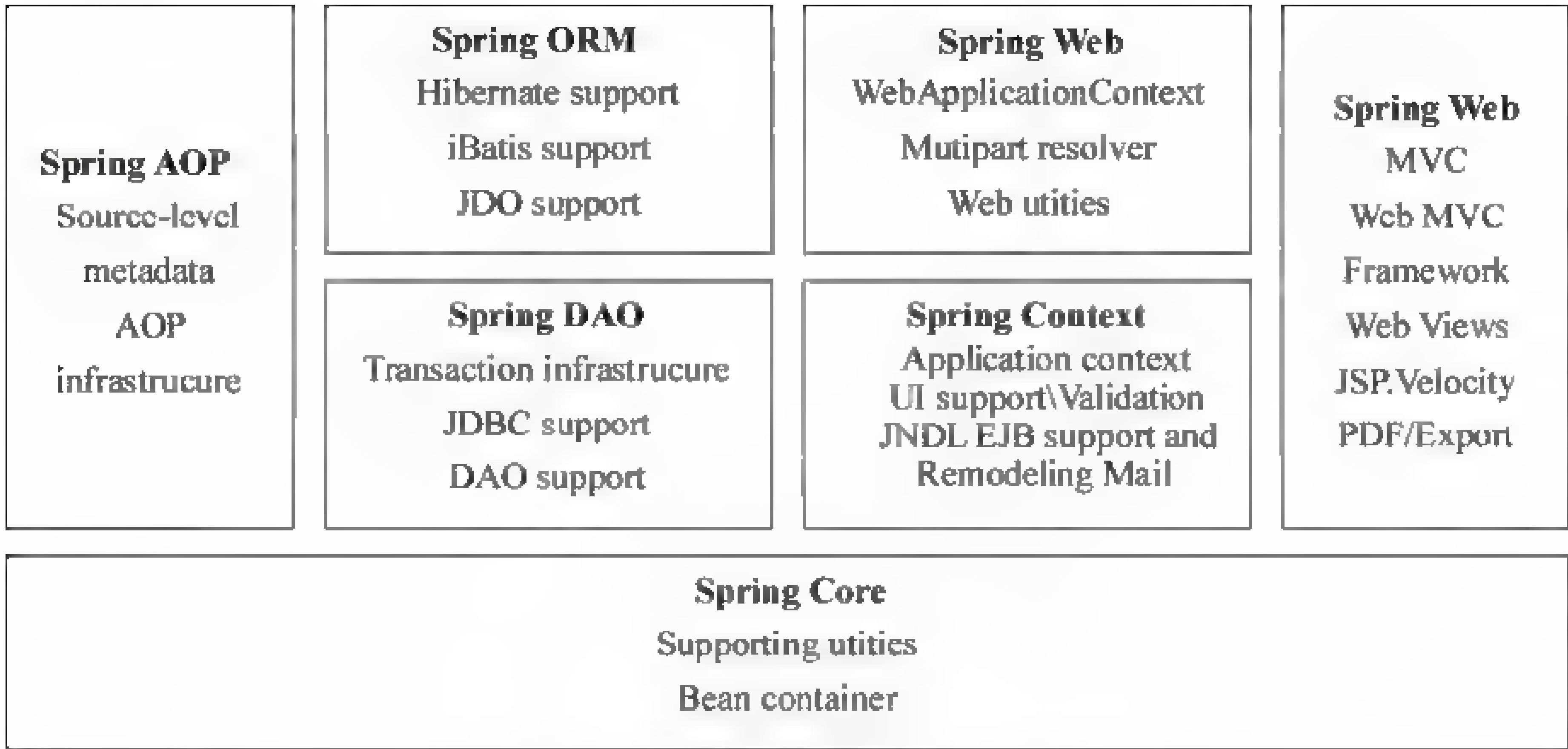

图 6-23　Spring 框架组成结构

组成 Spring 框架的每个模块（或组件）都可以单独存在，或者与其他一个或多个模块整合实现。并且 Spring 框架的功能可以用在任何 J2EE 服务器中，大多数功能也适用于不受管理的环境。Spring 的核心要点是：支持不绑定到特定 J2EE 服务的可重用业务和数据访问对象。这样的对象可以在不同 J2EE 环境（Web 或 EJB）、独立应用程序、测试环境之间重用。在图 6-23 所示中，Spring 框架的七个模块描述如下：

6.6.1.1　核心容器

核心容器提供 Spring 框架的基本功能，它提供了 IoC 功能，允许对 bean 容器进行管理，即该核心的一个基本组件是 BeanFactory，这是基本工厂模式的一个实现，并使应用程序的配置和依赖性规范与实际编程逻辑清晰地分离开。org.springframework.beans 包为 Spring 的 IoC 特性提供了基础，以及实现了 Bean 的定义、Bean 的创建和对 Bean 的解析。需要考虑的最重要的接口之一是 BeanFactory 接口，它有三个子类：ListableBeanFactory、HierarchicalBeanFactory 和 AutowireCapableBeanFactory。通过使用高级配置，BeanFactory 能够管理任何性质或复杂性的 bean。而 BeanFactory 是创建和管理应用程序所需的大量 bean 的一个容器，这些 bean 的性质可以有很大差异，并且有些只具有基本属性的简单 bean；有些则可以与其他 bean 协同工作，所以具有依赖性；还有些则具有递归的依赖性。BeanFactory 通过配置文件来管理这些依赖性。最常用的 BeanFactory 实现是 org.springframework.beans.factory.xml.XmlBeanFactory，程序片段如下：

```
Resource resource = new FileSystemResource("beans.xml");
XMLBeanFactory beanFactory = new XMLBeanFactory(resource);
//构造一个 BeanFactory 的实例
```

BeanFactory 配置中包括的 BeanFactory 经常要管理的一个或多个 bean 的定义。在 XmlBeanFactory 中，它们被配置为在顶级 bean 元素内的一个或多个 bean 元素，图 6-24 是创建 Bean 时序图。如下程序片段是 XmlBeanFactory 配置方法：

```
<beans>
   <bean id " " class " ">
```

```
        ...
    </bean>
      ...
</beans>
```

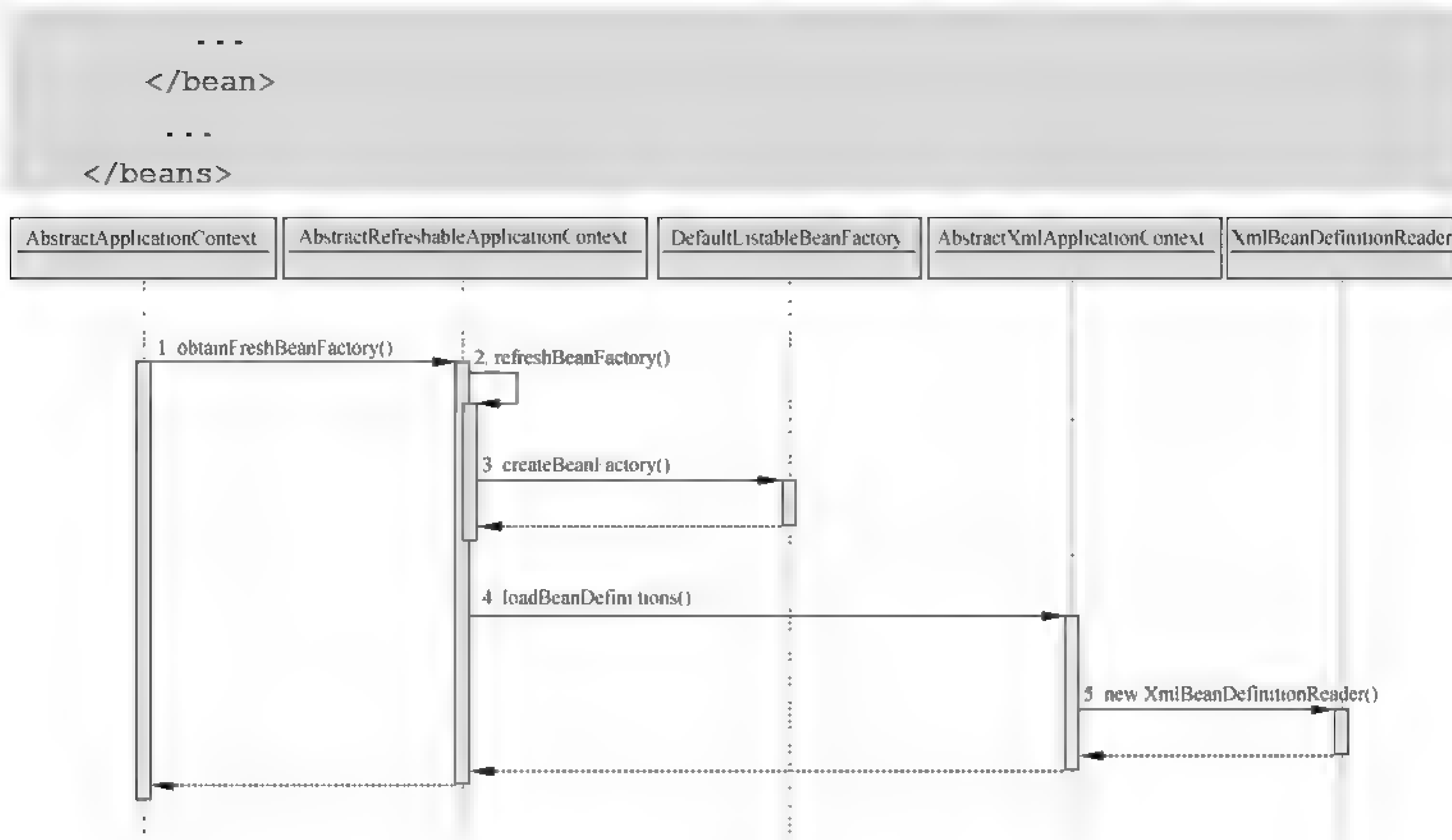

图 6-24　创建 Bean 的时序图

同时，建设 Bean 实例是从 finishBeanFactoryInitialization 方法开始的。如果一个类继承 FactoryBean 用户后，则可以自己定义产生实例对象的方法，这时只要实现它的 getObject 方法即可。然而在 Spring 内部的这个 Bean 实例对象是 FactoryBean，通过调用这个对象的 getObject 方法就能获取用户自定义产生的对象，从而为 Spring 提供了很好的扩展性。Spring 获取 FactoryBean 本身的对象是在前面加上&来完成的。图 6-25 是 Bean 创立的时序图。

6.6.1.2　Spring 上下文

Spring 上下文是一个配置文件，向 Spring 框架提供上下文信息。Spring 上下文包括企业服务，例如 JNDI、EJB、电子邮件、国际化、校验和调度功能。

BeanFactory 为应用程序提供了配置框架和基本功能，而 ApplicationContext 则为它添加了增强功能。由于 ApplicationContext 是 BeanFactory 的子类，所以它具有 BeanFactory 所提供的所有功能，并向其中添加了许多专有的特性。这些特性包括与 SpringAOP 特性轻松集成、消息资源处理、用于 i18n（国际化）、对资源（如 URL 和文件）的访问、事件处理和传播给实现 ApplicationListener 接口的 bean，以及透明地创建不同上下文的高级声明机制，如可选的父上下文和特定于应用程序层的上下文。

同时，ApplicationContext 构造是 BeanFactory 的一个完全超集，对 BeanFactory 功能的任何引用也应该同样适用于 ApplicationContext。在特定情形下，有时很难明确地决定该使用 BeanFactory 还是 ApplicationContext。由于 ApplicationContext 提供了 BeanFactory 的所有特性，而且在允许以更具说明性的方式使用一些功能的同时，还为它增加了另外一些特性，所以与 BeanFactory 比较而言，该类更优越一点。比如在内存中对于每千字节都很重要的 applet，使用 BeanFactory 是一个好的选择。图 6-26 是 Context 相关的类结构图。

AbstractApplicationContext
DefaultListableBeanFactory
AbstractBeanFactory
DefaultSingletonBeanRegistry
ObjectFactory
1. 调用preInstantiateSingletons方法
2. 调用getMergedLocalBeanDefinition
3. 调用getMergedBeanDefinition
4. 调用getBean
5. 调用doGetBean
6. 调用getSingleton
7. 调用getObject
8. createBean
对象工厂将会调用装配工厂，然后会调用初始化实例策略产生实例对象
9.
将创建的实例对象放入缓存对象中保存
10. 调用getObjectForBeanInstance
判断是否是FactoryBean

图 6-25　Bean 实例创建时序图

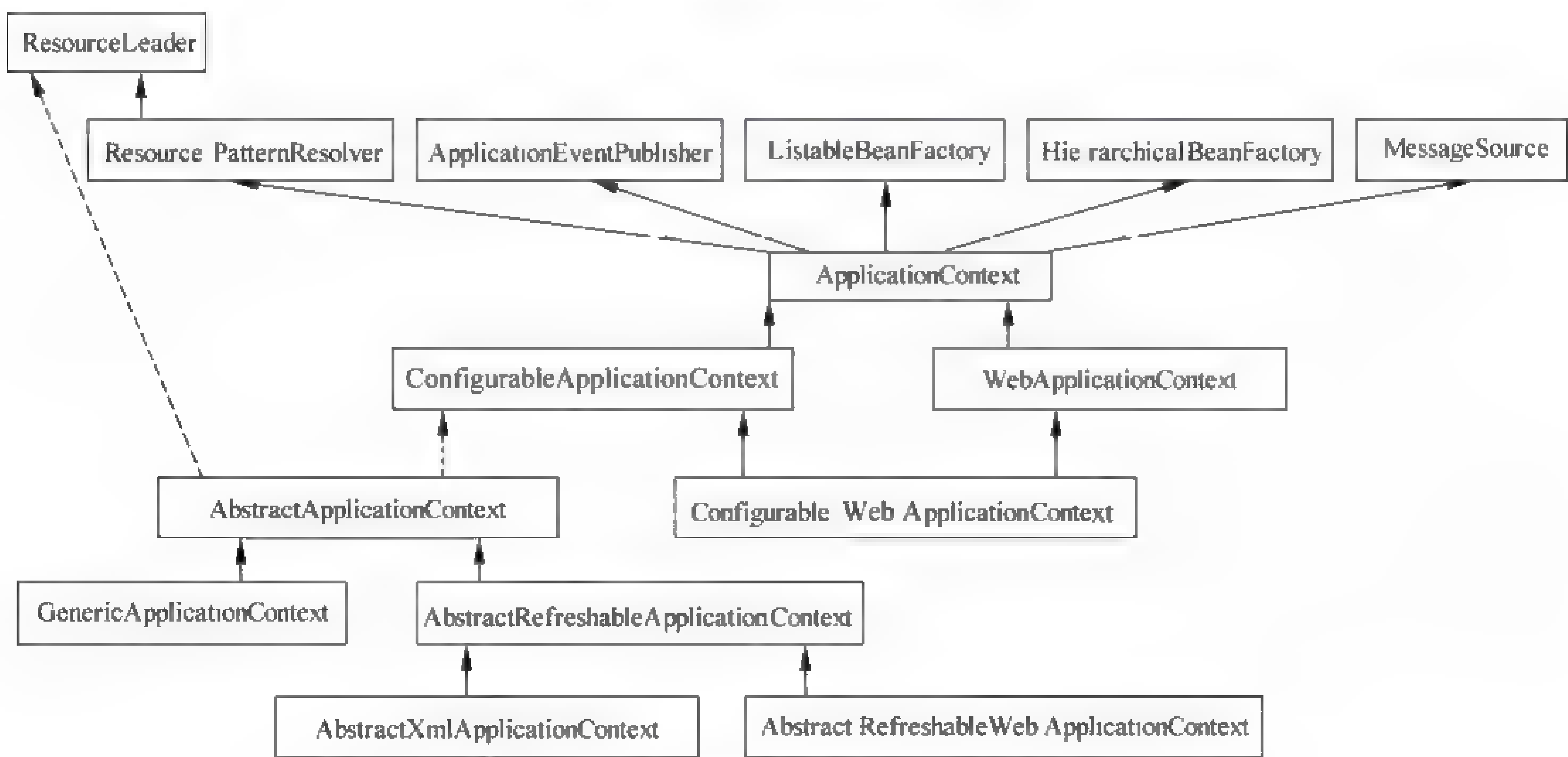

图 6-26　Context 相关的类结构图

6.6.1.3　Spring AOP

通过配置管理特性，Spring AOP 模块直接将面向方面的编程功能集成到了 Spring 框架中。所以，可以很容易地使 Spring 框架管理的任何对象支持 AOP。Spring AOP 模块为基于 Spring 的应用程序中的对象提供了事务管理服务。通过使用 Spring AOP，不用依赖 EJB 组件，就可以将声明性事务管理集成到应用程序中。

AOP 通过着眼于方面或关注点（Concern），而不是对象，这样就扩展了 OOP 的概念。AOP 应用程序按方面或关注点进行分解，否则，一个方面或关注点可能跨越多个对象。如事务和池（pooling）就是方面（aspect）的例子。

6.6.1.4　Spring DAO

JDBC DAO 抽象层提供了有意义的异常层次结构，可用该结构来管理异常处理和不同数据库供应商抛出的错误消息。异常层次结构简化了错误处理，并且极大地降低了需要编写的异常代码数量（例如打开和关闭连接）。Spring DAO 的面向 JDBC 的异常遵从通用的 DAO 异常层次结构。

Spring DAO 框架的主要目标是让数据访问技术（如 JDBC、Hibernate 或 JDO）相关的工作得以标准化和简化。如果使用 Spring DAO 框架，这就可以从一种数据访问技术转移到另一种会变得相当容易。而且这个 DAO 支持以下几个超类：

（1）JdbcDaoSupport：JDBC 数据访问对象的超类。需要设置一个 DataSource，向子类提供一个基于它的 JdbcTemplate。

（2）HibernateDaoSupport：Hibernate 数据访问对象的超类。需要设置一个 SessionFactory，向子类提供一个基于它的 HibernateTemplate。

（3）JdoDaoSupport：JDO 数据访问对象的超类。需要设置一个 PersistenceManagerFactory，向子类提供一个基于它的 JdoTemplate。

Spring 框架中的 JDBC 试图通过一个非常有特色的方式解决这些问题。简单地说，它将连接管理功能和其他数据库相关的资源管理功能进行抽象，不再由开发人员管理，这就使得资源的关闭更加正确，代码可读性也得到了提高。因此，Spring 所提供的 JDBC 抽象框架包括四个不同的包——核心包（包括 JdbcTemplate 类。它采取了 Web 应用程序中最常用的模板模式，它是 JDBC 核心包中最主要的类）、数据源包（包括一个用于简化 DataSource 访问的实用工具类，它还包括了各种用于测试数据库访问代码的 DataSource 实现）、对象包[包括一些类，它们把将关系数据库管理系统（RDBMS）的查询、更新和存储过程表示为线程安全的可重用对象]和支持包（包括许多实用工具类和 SQLException 翻译功能）。

6.6.1.5　Spring ORM

Spring 框架插入了若干个 ORM 框架，从而提供了 ORM 的对象关系工具，其中包括 JDO、Hibernate 和 iBatis SQL Map。所有这些都遵从 Spring 的通用事务和 DAO 异常层次结构。使用 Spring 创建 ORMDAO 的其他好处有：

（1）易于测试：如前所述，采用 Spring 的 IoC 方法，对于与对象关系有关的不同实体的实现和配置位置，可以很容易地进行切换。这样就很容易隔离地测试每一段与持久性有关的

代码。

（2）通用的数据访问异常：Spring 可以把选择的 ORM 工具抛出的异常包装为一组易于理解的、定义好的异常。

（3）集成的事务管理：Spring 不仅处理事务语义，对于回滚之类的操作，还完成适当的事务管理工作。

（4）避免供应商锁定并允许随意选用的实现策略：使用 Spring 的去耦方法，就有可能在运行时换用不同的 API 和实现。因此将不会锁定于使用某个供应商的产品和服务，而是可以根据需要随意选用。

正如前面所述：Spring 方便了资源管理、DAO 实现支持与几种 ORM 工具的事务策略集成，这些 ORM 工具的例子有：Hibernate、JDO、Oracle Top Link、Apache ObJectRelationalBridge (OJB)和 iBATIS SQL Maps。Spring 正如 JdbcTemplate 处理应用程序中大多数资源管理功能和执行顺序一样，它也提供了一个 HibernateTemplate 和 HibernateCallback，以便能与底层数据访问技术和事务技术实现清晰地隔离，从而使应用程序对象之间的耦合变得更为松散。

同时，为了避免应用程序对象与资源查找表紧密地关联起来，Spring 中允许把像 JDBC DataSource 或 Hibernate SessionFactory 这样的资源定义为一个应用程序上下文中的 bean。需要访问资源的应用程序对象只是通过对 bean 的引用来接受对这类预定义实例的引用。如下程序片段构造了一个 JDBC DataSource，并在其上构造一个 Hibernate SessionFactory。

```
<beans>
<bean id="myDataSource" class="org.apache.commons.dbcp.BasicDataSource"
destroy-method="close">
<property name="driverClassName" value="org.hsqldb.jdbcDriver"/>
<property name="url" value="jdbc:hsqldb:hsql://localhost:8080"/>
<property name="username" value="  "/>
<property name="password" value="  "/>
</bean>
<bean id="mySessionFactory"
 class="org.springframework.orm.hibernate.LocalSessionFactoryBean">
<property name="dataSource" ref="myDataSource"/>
<property name="mappingResources">
<list>
<value>product.hbm.xml</value>
</list>
</property>
<property name="hibernateProperties">
<props>
<prop key="hibernate.dialect">
net.sf.hibernate.dialect.MySQLDialect
</prop>
</props>
</property>
</bean>
...
</beans>
```

同样，应用程序对象的实现只需要有一个 Hibernate SessionFactory，它就可以通过 Spring 应用程序上下文的一个简单 bean 引用来提供。如下程序片段就是在 Spring 应用程序上下文中定义一个 DAO，其中引用了前面定义的 SessionFactory。

```
<beans>
...
<bean id="employeeDao" class="emp.EmployeeDaoImpl">
<property name="sessionFactory" ref="mySessionFactory"/>
</bean>
</beans>
```

6.6.1.6　Spring Web 模块

Web 上下文模块建立在应用程序上下文模块之上，为基于 Web 的应用程序提供了上下文。所以，Spring 框架支持与 Struts 的集成。Web 模块还简化了处理多部分请求，以及将请求参数绑定到域对象的工作。Spring Web 流引擎对第三方 API 的依赖性非常小，而且所有的依赖性都得到仔细的管理。并且它可以清楚简单地显示 Web 应用程序的页面流，也可以任何地方重用它，包括像 Struts、Spring MVC、Tapestry、JavaServer Faces (JSF) ，甚至是 portlet 这样的环境。即 Spring Web 是一个基于有限状态机的功能强大的控制器，它完全解决了 MVC 结构中 C（控制）的问题。

6.6.1.7　Spring MVC 框架

MVC 框架是一个全功能的构建 Web 应用程序的 MVC 实现。通过策略接口、MVC 框架变成为高度可配置。MVC 容纳了大量视图技术，其中包括 JSP、Velocity、Tiles、iText 和 POI。

Spring 框架为构建 Web 应用程序提供了一个 MVC 框架。Spring 的 MVC Web 应用程序框架构建在核心功能之上。它是一个与 Struts 类似的基于请求的框架，但试图解决 Struts 所暴露出来的缺陷。对于现代基于请求的框架必须处理的所有职能，Spring MVC 框架定义了不同的策略接口，这些接口的职能都很简单和清楚，所以，Spring MVC 用户可以很容易地编写自己的实现。它是围绕 DispatcherServlet 设计的，这个组件具有前端控制器的职能，在 HTTP 请求的执行阶段，它负责把控制委派给各个接口。默认处理器是一个非常简单的控制器接口，它只有一个方法：ModelAndView handleRequest(request, response)。

Spring MVC 所提供的高度抽象最重要的优点之一是：测试这些接口和整个应用程序的实现变得非常容易。DispatcherServlet 的设计允许开发人员以一种简单而一致的方式按照 Spring IoC 模型来配置应用程序的 Web 层。下面是 Spring MVC 定义的最重要的接口：

（1）HandlerMappings：通过使用处理程序映射，可以把传入的请求映射到适当的处理程序。Spring MVC 还提供了一组前置和后置处理器和控制器，它们在特定条件下执行，例如匹配指定的 URL 和控制器。

（2）HandlerAdapater：运行一个适当的对象来处理收到的请求。

（3）Controller(s)：这些 bean 提供处理传入请求的实际功能，也就是 MVC 中的 C。

（4）View Resolver：能够把视图名解析为视图。

（5）Locale Resolver：能够解析客户机用于 i18n 支持的区域。

（6）Theme Resolver：如果应用程序提供基于主题的个性化视图，该对象可以解析这些主题。

（7）Multipart Resolver：提供了处理 HTML 表单的多部分文件上载的功能。

Spring 是一个功能强大的框架，它试图解决 J2EE 和 Java EE 开发中存在的普遍问题。它还非常灵活，可用于 J2EE 和 Java EE 之外的环境。它的目标是让大家记住面向对象编程中正确的做法，即使用接口来设计应用程序。

6.6.2 AOP

面向方面的编程（Aspect-Oriented Programming，AOP）是一种编程技术，它允许程序员对横切关注点或横切典型的职责分界线的行为（例如日志和事务管理）进行模块化。AOP 的核心构造是方面，它将那些影响多个类的行为封装到可重用的模块中。在典型的面向对象开发方式中，可能要将日志记录语句放在所有方法和 Java 类中才能实现日志功能。在 AOP 方式中，可以反过来将日志服务模块化，并以声明的方式将它们应用到需要日志的组件上。当然，优势就是 Java 类不需要知道日志服务的存在，也不需要考虑相关的代码。所以，用 Spring AOP 编写的应用程序代码是松散耦合的。

AOP 主要关注横切，是面向方面编程的专有名词，指的是在一个给定的编程模型中穿越既定的职责部分（比如日志记录和性能优化）的操作。在横切的世界里，横切有两种类型：动态横切和静态横切。如下程序片段就是方面的配置方法：

```
<beans>
    <bean name="RemoteExceptionHandlingAspect"
      class="org.aspectprogrammer.dw.RemoteExceptionHandling"
      factory-method="aspectOf">
      <property name="exceptionHandler">
        <ref bean="RemoteExceptionHandler"/>
      </property>
    </bean>
    <bean name="RemoteExceptionHandler"
      class="org.aspectprogrammer.dw.DefaultRemoteExceptionHandler">
    </bean>
  </beans>
```

动态横切是通过切入点和连接点在一个方面中创建行为的过程，连接点可以在执行时横向地应用于现有对象。动态横切通常用于帮助向对象层次中的各种方法添加日志记录或身份认证。主要由以下几点来组成动态横切：

（1）方面（aspect）类似于 Java 编程语言中的类。方面定义切入点和通知（advice），并由诸如 AspectJ 这样的方面编译器来编译，以便将横切（包括动态的和静态的）织入（interweave）现有的对象中。

（2）一个连接点（join point）是程序执行中一个精确执行点，比如类中的一个方法。例如，对象 Foo 中的方法 bar()就可以是一个连接点。连接点是个抽象的概念，不用主动定义一

个连接点。

（3）一个切入点（pointcut）本质上一个用于捕捉连接点的结构。例如，可以定义一个切入点来捕捉对对象 Foo 中的方法 bar()的所有调用。和连接点相反，切入点需要在方面中定义。

（4）通知（advice）是切入点的可执行代码。一个经常定义的通知是添加日志记录功能，其中切入点捕捉对对象 Foo 中的 bar()的每个调用，然后该通知动态地插入一些日志记录功能，比如捕捉 bar()的参数。

这其中，核心就是切入点（pointcut）和通知（advice）的声明。切入点描述了主程序执行与方面执行相遇的地方，也就是被横切的位置；通知则描述了在程序执行过程中遇到匹配的切入点时应当采取什么行动。假设已经开发了一个方面，并且感觉它适用于其他项目，那么可以泛化这个方面，并把它隔离到独立的项目中，形成一个库，即方面库（Aspect Library），该库提供了某个功能的内部执行逻辑和基础设施，通过切入点的实例化将方面库与某个特定项目连接起来。例如提供应用程序性能监视的方面库，实现了所有性能监视相关的方法和通知，当某应用程序使用该库的时候，只需要把库的切入点定义为应用特定的连接点即可，而无需关心性能监视功能的具体实现。

而静态横切和动态横切的区别在于它不修改一个给定对象的执行行为。相反，它允许通过引入附加的方法字段和属性来修改对象的结构。此外，静态横切可以把扩展和实现附加到对象的基本结构中。

针对 AOP 的横切实现，AspectJ 是一个面向方面的框架，它扩展了 Java 语言。AspectJ 定义了 AOP 语法所以它有一个专门的编译器用来生成遵守 Java 字节编码规范的 Class 文件。因此，AspectJ 针对方面库的实现在于切入点的实例化方式。方面在 AspectJ 语言中用 aspect 关键字标示，它类似于 Java 类，可以定义成抽象的，也存在继承关系。切入点在 AspectJ 语言中用关键字 pointcut 标示，它有一套完整的语法来描述切入点，也可以定义成抽象的，即没有实际定义的切入点。抽象 pointcut 只能定义在抽象方面中，如下程序片段所示。抽象方面 A 里面定义了一个抽象 pointcut 名叫 publicCall。同时，抽象方面如同抽象类一样，可以被继承，继承抽象方面的方面必须重载抽象 pointcut，即赋予抽象 pointcut 实际的定义。AspectJ 对切入点定义方法的支持导致了两种完全不同的方面库实现方法：使用抽象方面（abstract aspect）的方法和使用注释的方法。

下面是 AOP 的程序片段：

```
public abstract aspect A {
     abstract pointcut publicCall(int i);
  }
//抽象方面
public aspect B extends A {
     pointcut publicCall(int i): call(public Foo.m(int)) && args(i);
}
//继承抽象方面的子方面
```

AspectJ 对继承和抽象的支持正是构造方面库的基础。抽象方面包含抽象的切入点和具体的通知，正符合方面库的特征，可以使用抽象方面来构造方面库文件。继承抽象方面的子方面必须具体化切入点，可以把它当作方面库在具体应用程序中的实施。因此，从 AspectJ 5

开始支持的注释是另外一种构造方面库的技术基础。AspectJ 5 能支持的注释包括修饰方面、方法、属性、构造函数和通知，修饰方法和通知的参数的注释也能支持，但是不支持 pointcut 和 declare 语句上的注释。为了支持注释类型，AspectJ 5 扩展了 pointcut 语法，可以匹配存在或者不存在的注释类型。如下程序片段中的名叫 onewayMethod 的 pointcut 可以匹配所有被注释@Oneway 修饰的方法调用。

```
public aspect C {
pointcut onewayMethod: call(@Oneway * *(..));
}
//含有注释的 pointcut
```

AspectJ 5 对注释的支持简化了实施库的方法，由此可以利用注释标明具体的切入点的位置。在构造方面库文件时，只需要定义好与注释相关的切入点，并规定该切入点上的具体的通知内容就可以了。

AspectJ 拥有支持装入时织入必须的基础设施，但是必须编写定制的类装入器，才能真正把 AspectJ 的织入器集成到应用程序。在 AspectJ 5 中，通过配置文件 META-INF/aop.xml 来支持对装入时织入配置。程序片段如下：

```
<aspectj>
  <aspects>
      <!-- declare two existing aspects to the weaver -->
      <aspect name="com.MyAspect"/>
      <aspect name="com.MyAspect.Inner"/>
      <!-- define a concrete aspect inline -->
      <concrete-aspect name="com.xyz.tracing.MyTracing"
       extends="tracing.AbstractTracing">
        <pointcut name="tracingScope" expression="within(com.xyz..*)"/>
      </concrete-aspect>
  </aspects>
  <weaver options="-XlazyTjp">
      <include within="com.xyz..*"/>
  </weaver>
 </aspectj>
```

在 Spring 中实现 AOP 时，AOP 是基于动态代理实现的。动态代理还要从 JDK 下的 java.lang.reflect 包下的 Proxy 类来实现，它正是构造代理类的入口[①]。如下程序片段是一个新代理类实现方法：

```
public static Object newProxyInstance(ClassLoader loader, Class<?>[] interfaces,
InvocationHandler h)
    throws IllegalArgumentException {

     if (h == null) {
```

① http://www.ibm.com/developerworks/cn/java/j-lo-spring-principle/index.html?ca=drs-

```
            throw new NullPointerException();
        }
        Class cl = getProxyClass(loader, interfaces);
        try {
            Constructor cons = cl.getConstructor(constructorParams);
            return (Object) cons.newInstance(new Object[] { h });
        } catch (NoSuchMethodException e) {
            throw new InternalError(e.toString());
        } catch (IllegalAccessException e) {
            throw new InternalError(e.toString());
        } catch (InstantiationException e) {
            throw new InternalError(e.toString());
        } catch (InvocationTargetException e) {
            throw new InternalError(e.toString());
        }
    }
```

在程序片段中，方法需要三个参数：ClassLoader，用于加载代理类的 Loader 类，通常这个 Loader 和被代理的类是同一个 Loader 类。Interfaces 是要被代理的那些接口。InvocationHandler 是用于执行除了被代理接口中方法之外的用户自定义的操作，也是用户需要代理的最终目的。用户调用目标方法都被代理到 InvocationHandler 类中定义的唯一方法 invoke 中。图 6-27 是创建代理对象时序图。

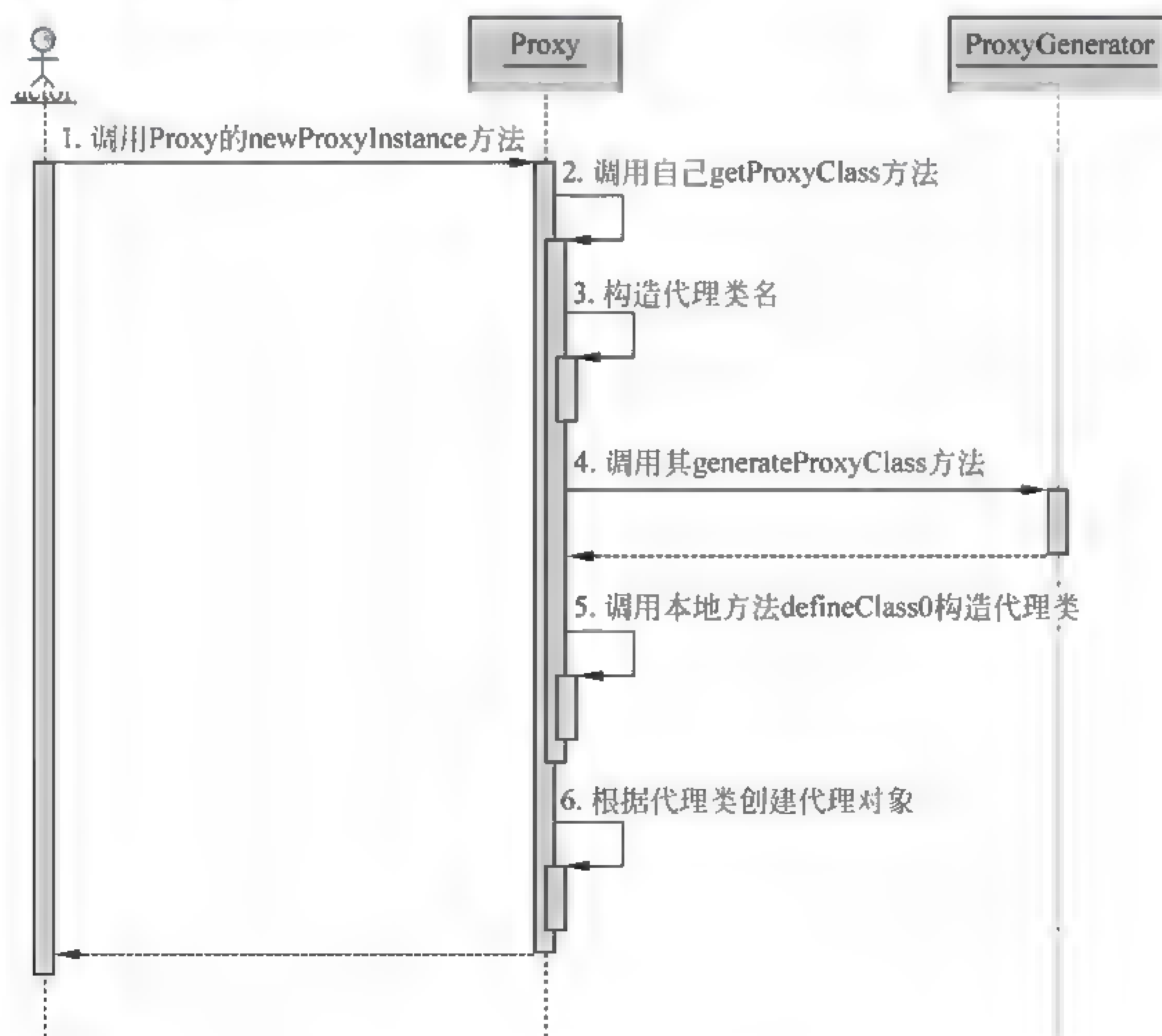

图 6-27　创建代理对象时序图

代理的目的是调用目标方法时可以转而执行 InvocationHandler 类的 invoke 方法，所以如何在 InvocationHandler 上做实现代理，就是 Spring 实现 Aop 的关键所在。Spring 的 Aop 实现是遵守 Aop 联盟的约定。但要实现的代理类在 Spring 的配置文件中通常是这样定义一个 Bean 的，程序片段如下：

```
<bean id="testBeanSingleton"
   class="org.springframework.aop.framework.ProxyFactoryBean">
   <property name="proxyInterfaces">
      <value>
         org.springframework.aop.framework.PrototypeTargetTests$TestBean
      </value>
   </property>
   <property name="target"><ref local="testBeanTarget"></ref> </property>
   <property name="singleton"><value>true</value></property>
   <property name="interceptorNames">
      <list>
         <value>testInterceptor</value>
         <value>testInterceptor2</value>
      </list>
   </property>
</bean>
```

Spring Aop 实现了自身的扩展点来完成拦截特性的，从这个代理类可以看出它正是继承了 FactoryBean 的 ProxyFactoryBean，FactoryBean 之所以特别就在它可以让自定义对象的创建方法。当然代理对象要通过 Proxy 类来动态生成。图 6-28 所示是 Spring 代理对象的产生。

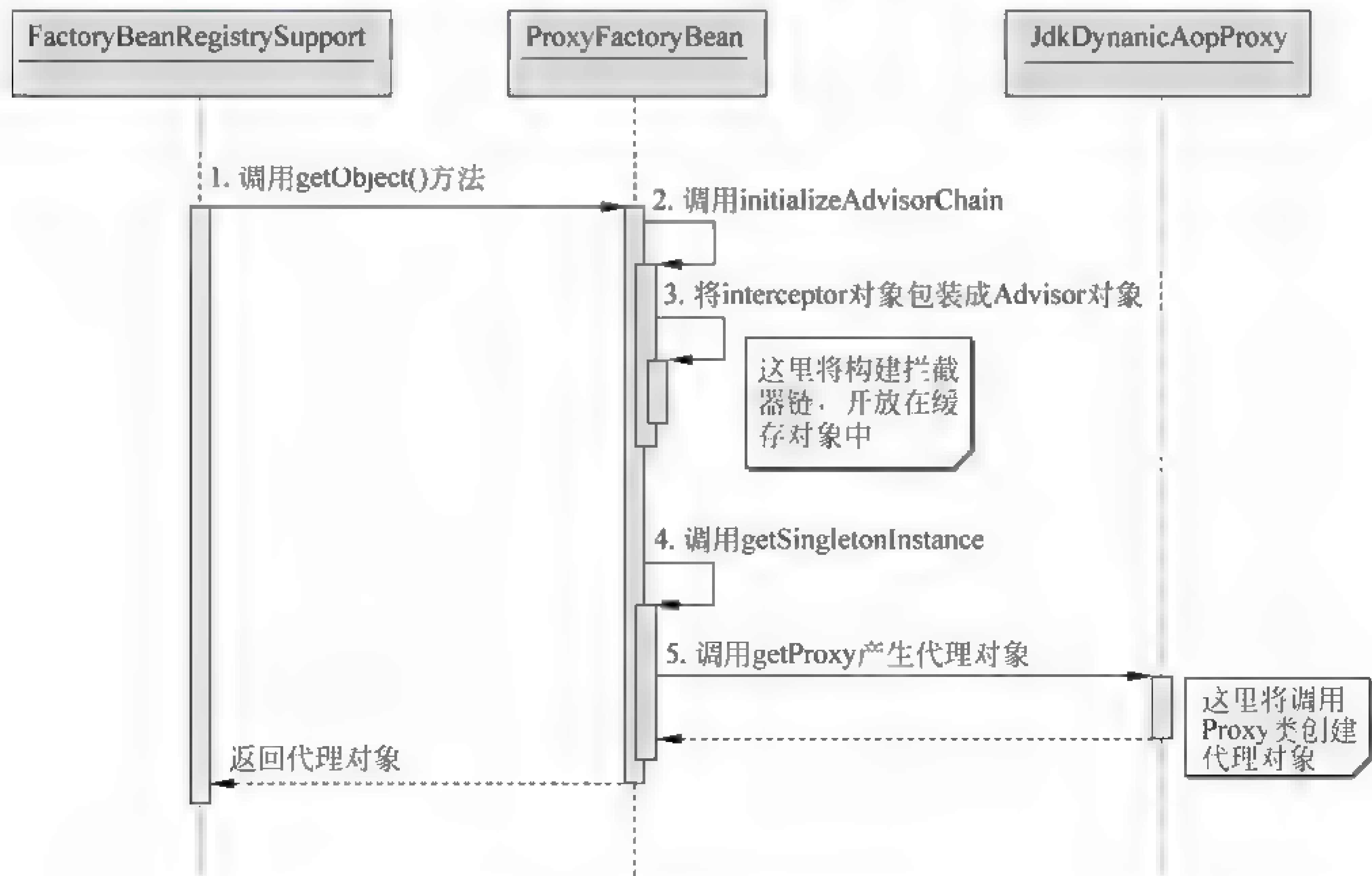

图 6-28　Spring 代理对象的产生

Spring 创建了代理对象后，当调用目标对象上的方法时，将都会被代理到 InvocationHandler 类的 invoke 方法中执行。在这里 JdkDynamicAopProxy 类实现了 InvocationHandler 接口。图 6-29 是 Spring 调用拦截器序列图。

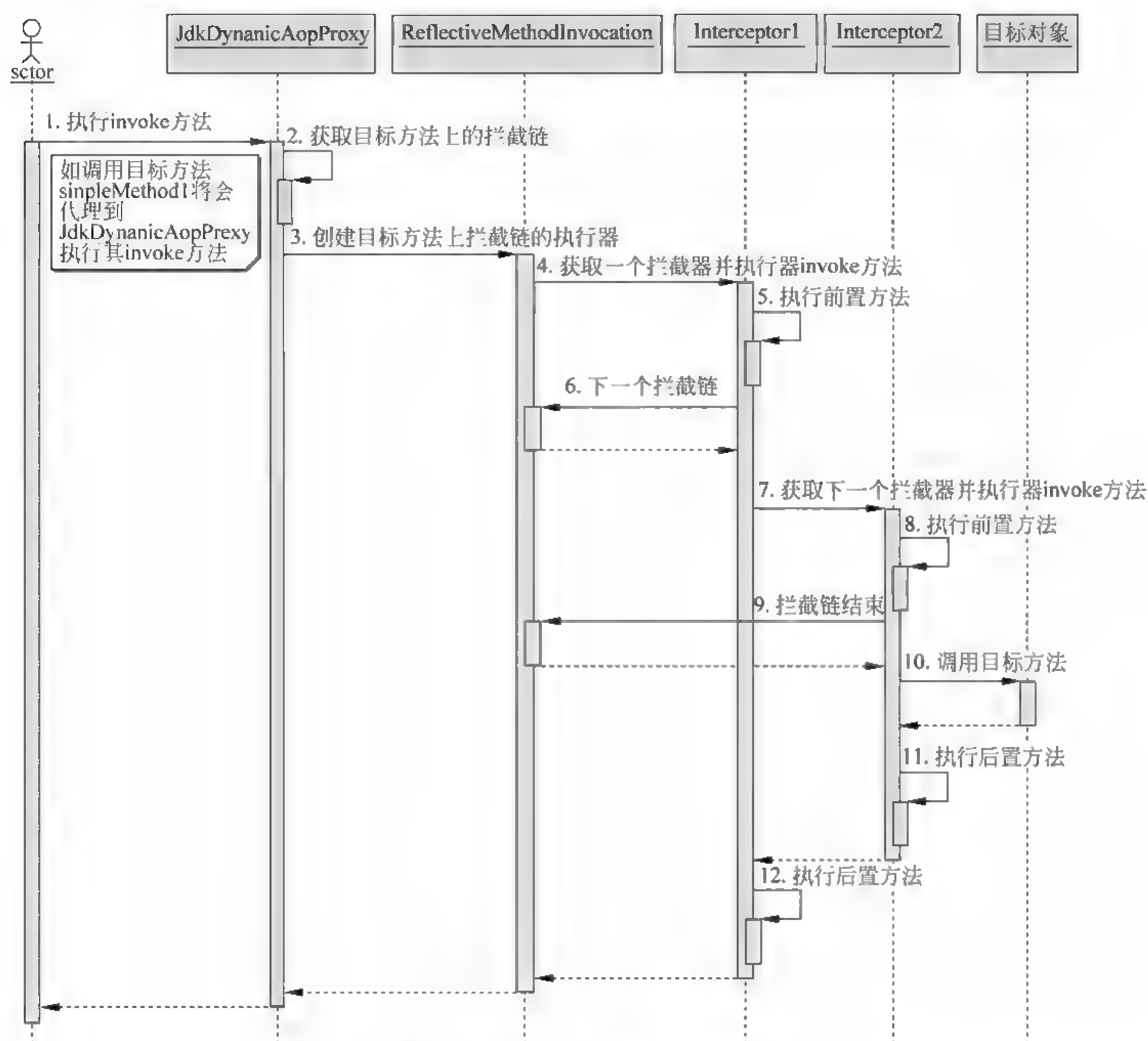

图 6-29 Spring 调用拦截器序列图

6.6.3 IoC

Ioc（Inversion Of Control）容器实际上就是 Context 组件结合其他两个组件共同构建了一个 Bean 关系网。是推动 Spring 框架发展的重要的组成部分，即它和 AOP 一起构成了 Spring 的核心，并且这种推动是通过基于容器的配置实现的。过去，Spring 允许开发人员使用基于 XML 的配置，通过利用应用程序上下文 XML 文件来管理 bean 依赖性。此文件处于应用程序的外部，包含 bean 及其与该应用程序的依赖项的定义。尽管使用 XML 配置较为简单和便捷，但仍有另外一种方法可定义 bean 及其依赖项。这种方法又称基于 Java 的配置。不同于 XML，基于 Java 的配置能够以编程方式管理 bean。这可通过运用多种注释来实现。正如在

6.6.2 小节中所叙述一样，Spring 设计的核心是 org.springframework.beans 包，它的设计目标是与 JavaBean 组件一起使用。这个包通常不是由用户直接使用，而是由服务器将其用作其他多数功能的底层中介。下一个最高级抽象是 BeanFactory 接口，它是工厂设计模式的实现，允许通过名称创建和检索对象。BeanFactory 也可以管理对象之间的关系。又因为 org.springframework.beans.factory.BeanFactory 是一个简单接口，所以可以针对各种底层存储方法实现。最常用的 BeanFactory 定义是 XmlBeanFactory，它根据 XML 文件中的定义装入 bean，程序片段如下：

```
BeanFactory factory = new XMLBeanFactory(new FileInputSteam("mybean.xml"))
```

在 XML 文件中定义的 Bean 是被消极加载的，这意味在需要 bean 之前，bean 本身不会被初始化。要从 BeanFactory 检索 bean，只需调用 getBean()方法，传入将要检索的 bean 的名称即可，程序片段如下：

```
MyBean mybean = (MyBean) factory.getBean("mybean")
```

这样，每个 bean 的定义都可以是 POJO 或 FactoryBean，FactoryBean 接口为使用 Spring 框架构建的应用程序添加了一个间接的级别。然而，对 Spring 的 Ioc 容器来说，主要有 BeanFactoryPostProcessor 和 BeanPostProcessor 两个接口，它们分别是在构建 BeanFactory 和构建 Bean 对象时调用。还有就是 InitializingBean 和 DisposableBean，它们分别是在 Bean 实例创建和销毁时被调用。用户可以实现这些接口中定义的方法，Spring 就会在适当的时候调用它们。还有一个是 FactoryBean，它是个特殊的 Bean，这个 Bean 可以被用户更多的控制。

有时在应用表达上，IoC 又称为依赖注入。Spring 既负责创建 bean，也负责配置 bean。但是，AspectJ 方面是由 AspectJ 运行时创建的。因此，依赖注入主要是以 bean 的操作。同时，Spring 的依赖配置方式与 Spring 框架的内核自身是松耦合设计的。现从以下几个方面进行叙述 Spring3.0 以后版本的依赖注入的基本方法。

6.6.3.1 使用@Configuration 和@Bean 进行 Bean 的声明

在将 bean 定义为基于 Java 的配置的一部分时。AppContext 类就像 XML 一样表示配置类。这是通过利用 @Configuration 注释实现的，@Configuration 注释位于类的顶端。它告知 Spring 容器这个类是一个拥有 bean 定义和依赖项的配置类。而@Bean 注释用于定义 bean。如下程序片段，它叙述了注释实例化 bean 并设置依赖项的方法上方。方法名称与 bean id 或默认名称相同。该方法的返回类型是向 Spring 应用程序上下文注册的 bean。这样就可使用 bean 的 setter 方法来设置依赖项，容器将调用它们来连接相关项。则基于 Java 的配置也被视为基于注释的配置。

```
@Configuration
public class AppContext {
    @Bean
    public Course course() {
        Course course = new Course();
        course.setModule(module());
        return course;
```

```
    }
    @Bean
    public Module module() {
        Module module = new Module();
        module.setAssignment(assignment());
        return module;
    }
    @Bean
    public Assignment assignment() {
        return new Assignment();
    }
}
```

同时，在传统 XML 方法中，仍然可使用 ClassPathXmlApplicationContext 类来加载外部 XML 上下文文件。但在使用基于 Java 的配置时，有一个 AnnotationConfigApplicationContext 类。它是 ApplicationContext 接口的一个实现，能够注册所注释的配置类。此处的配置类是使用@Configuration 注释声明的 AppContext。在注册了所述类之后，@Bean 注释的方法返回的所有 bean 类型也会得到注册，程序片段如下：

```
public static void main(String[] args) {
  ApplicationContext ctx = new AnnotationConfigApplicationContext(AppContext.
  class);
  Course course = ctx.getBean(Course.class);
  course.getName();
}
```

在程序片段中，AppContext 配置类的注册方式是将其传递给 AnnotationConfigApplication-Context 构造函数。此外，还可以使用所述上下文类的 register 方法来注册配置类，程序片段如下：

```
public static void main(String[] args) {
  ApplicationContext ctx = new AnnotationConfigApplicationContext();
  ctx.register(AppContext.class)
}
```

这时，注册配置类将自动注册@Bean 注释的方法名称，因而其对应的 bean 就是 Course、Module 和 Assignment。随后就可以使用 getBean 方法来获取相关的 bean，并调用其业务方法。

6.6.3.2 使用@Repository、@Service、@Controller 和 @Component 将类标识为 Bean

@Repository 注解用于将数据访问层（DAO 层）的类标识为 Spring Bean，具体只须将该注解标注在 DAO 类上即可。

@Repository 注解，需要在 XML 配置文件中启用 Bean 的自动扫描功能，可以通过 <context:component-scan/>实现，程序片段如下：

```
@Repository
```

```
public class UserDaoImpl implements UserDao{ ... }
<beans ... >
   ...
<context:component-scan base-package="*.dao" />
...
</beans>
```

Spring 2.5 在@Repository 的基础上增加了功能类似额外三个注解：@Component、@Service、@Constroller。

（1）@Component 是一个泛化的概念，仅仅表示一个组件（Bean），可以作用在任何层次。

（2）@Service 通常作用在业务层，但是目前该功能与@Component 相同。

（3）@Constroller 通常作用在控制层，但是目前该功能与@Component 相同。

通过在类上使用@Repository、@Component、@Service 和@Constroller 注解，Spring 会自动创建相应的 BeanDefinition 象，并注册到 ApplicationContext 中。这些类就成了 Spring 受管组件，这三个注解除了作用于不同软件层次的类，并且使用方式与@Repository 是完全相同的。

当一个 Bean 被自动检测到时，会根据扫描器的 BeanNameGenerator 策略生成它的 bean 名称，程序片段如下：

```
<beans ...>
 <context:component-scan
    base-package="  " name-generator="a.SimpleNameGenerator"/>
 </beans>
```

默认情况下，对于包含 name 属性的@Component、@Repository、@Service 和@Controller，会把 name 取值作为 Bean 的名字。如果这个注解不包含 name 值或是其他被自定义过滤器是被发现的组件，默认 Bean 名称会是小写开头的非限定类名。如果不想使用默认 bean 命名策略，可以提供一个自定义的命名策略。

6.6.3.3 使用@PostConstruct 和@PreDestroy 指定生命周期回调方法

Spring Bean 是受 Spring IoC 容器管理，由容器进行初始化和释放的。即由 Spring 经过构造函数或者工厂方法创建的 Bean，就是已经初始化完成并立即可用的，而 Bean 无法自动完成释放，这时就提供了@PostConstruct 和@PreDestroy 依赖注入方法来完成此项工作。因此，Spring 1.x 为此提供了两种方式供用户指定执行生命周期回调的方法，从而完成 Bean 释放。

第一种方式是实现 Spring 提供的两个接口：InitializingBean 和 DisposableBean。如果希望在 Bean 初始化完成之后执行一些自定义操作，则可以让 Bean 实现 InitializingBean 接口，该接口包含一个 afterPropertiesSet()方法，容器在为该 Bean 设置了属性之后，将自动调用该方法。如果 Bean 实现了 DisposableBean 接口，则容器在销毁该 Bean 之前，将调用该接口的 destroy()方法。这种方式的缺点是，让 Bean 类实现 Spring 提供的接口，增加了代码与 Spring 框架的耦合度，因此不推荐使用。

第二种方式是在 XML 文件中使用<bean>的 init-method 和 destroy-method 属性指定初始

化之后和销毁之前的回调方法，代码无须实现任何接口。这两个属性的取值是相应 Bean 类中的初始化和销毁方法，方法名任意，但是方法不能有参数，程序片段如下：

```
<bean id="" class="" init-method="init" destroy-method="destroy">
    …
</bean>
```

6.6.3.4　@Required 进行 Bean 的依赖检查

依赖检查的作用是判断给定 Bean 的相应 Setter 方法是否都在实例化的时候被调用了，而不是判断字段是否已经存在值了。Spring 进行依赖检查时，只会判断属性是否使用了 Setter 注入。如果某个属性没有使用 Setter 注入，即使是通过构造函数已经为该属性注入了值，Spring 仍然认为它没有执行注入，从而抛出异常。另外，Spring 只管是否通过 Setter 执行了注入，而对注入的值却没有任何要求，即使注入的 <null/>，Spring 也认为是执行了依赖注入。

<bean>标签提供了 dependency-check 属性用于进行依赖检查。该属性的取值包括以下几种：

（1）None：默认不执行依赖检查。可以在 <beans> 标签上使用 default-dependency-check 属性改变默认值。

（2）simple：对原始基本类型和集合类型进行检查。

（3）Objects：对复杂类型进行检查（除了 simple 所检查类型之外的其他类型）。

（4）All：对所有类型进行检查。

为了让 Spring 能够处理该注解，需要激活相应的 Bean 后处理器。要激活该处理器，只需在 XML 中增加如下一行即可：

```
<context:annotation-config/>
```

6.6.3.5　使用@Resource、@Autowired 和@Qualifier 指定 Bean 的自动组装策略

自动组装是指 Spring 在组装 Bean 的时候，根据指定的自动组装规则，将某个 Bean 所需要引用类型的 Bean 注入进来。并且<bean>元素提供了一个指定自动装配类型的 autowire 属性，该属性有如下选项：

（1）no：显式指定不使用自动装配。

（2）byName：如果存在一个和当前属性名字一致的 Bean，则使用该 Bean 进行注入。如果名称匹配但是类型不匹配，则抛出异常。如果没有匹配的类型，则什么也不做。

（3）byType：如果存在一个和当前属性类型一致的 Bean（相同类型或者子类型），则使用该 Bean 进行注入。byType 能够识别工厂方法，即能够识别 factory-method 的返回类型。如果存在多个类型一致的 Bean，则抛出异常。如果没有匹配的类型，则什么也不做。

（4）constructor：与 byType 类似，只不过它是针对构造函数注入而言的。如果当前没有与构造函数的参数类型匹配的 Bean，则抛出异常。使用该种装配模式时，优先匹配参数最多的构造函数。

（5）autodetect：根据 Bean 的自省机制决定采用 byType 还是 constructor 进行自动装配。

如果 Bean 提供了默认的构造函数，则采用 byType；否则采用 constructor 进行自动装配。

当使用 byType 或者 constructor 类型的自动装配的时候，自动装配也支持引用类型的数组或者使用了泛型的集合，这样，Spring 就会检查容器中所有类型匹配的 Bean，组成集合或者数组后执行注入。对于使用了泛型的 Map 类型，如果键是 String 类型，则 Spring 也会自动执行装配，将所有类型匹配的 Bean 作为值，Bean 的名字作为键。

使用@Autowired 注解进行装配，只能是根据类型进行匹配。@Autowired 注解可以用于 Setter 方法、构造函数、字段，甚至普通方法，前提是方法必须有至少一个参数。@Autowired 可以用于数组和使用泛型的集合类型。然后 Spring 会将容器中所有类型符合的 Bean 注入进来。@Autowired 标注作用于 Map 类型时，如果 Map 的 key 为 String 类型，则 Spring 会将容器中所有类型符合 Map 的 value 对应的类型的 Bean 增加进来，用 Bean 的 id 或 name 作为 Map 的 key。

当标注了@Autowired 后，自动注入不能满足，则会抛出异常。这时可以给 @Autowired 标注增加一个 required=false 属性，以改变这个行为，程序片段如下：

```
@Autowired(required=false);
@Qualifier("ppp");
public void setPerson(person p){};
```

@Qualifier 甚至可以作用于方法的参数，程序片段如下：

```
@Autowired(required=false)
 public void sayHello(@Qualifier("ppp")Person p,String name){}
...
<bean id="person" class="footmark.spring.Person">
    <qualifier value="ppp"/>
</bean>
//在配置文件中指定某个 Bean 的 qualifier 名字
```

另外，每一个类中只能有一个构造函数的@Autowired.required()属性为 true，否则就出问题了。如果用@Autowired 同时标注了多个构造函数，那么，Spring 将采用贪心算法匹配构造函数。@Autowired 还有一个作用就是如果将其标注在 BeanFactory 类型、ApplicationContext 类型、ResourceLoader 类型、ApplicationEventPublisher 类型、MessageSource 类型上，那么 Spring 会自动注入这些实现类的实例，不需要额外的操作。当容器中存在多个 Bean 的类型与需要注入的相同时，注入将不能执行，这时可以给@Autowired 增加一个候选值，即在@Autowired 后面增加一个@Qualifier 标注，提供一个 String 类型的值作为候选的 Bean 的名字。

最后，配置文件中需要指定每一个自定义注解的属性值。这时可以使用 <meta> 标签来代替<qualifier/>标签，如果<meta>标签和<qualifier/>标签同时出现，那么优先使用<qualifier>标签。如果没有<qualifier>标签，那么会用<meta>提供的键值对来封装<qualifier>标签，程序片段如下：

```
<bean class="footmark.HelloWorld">
 <qualifier type="MovieQualifier">
 <attribute key="format" value="VHS"/>
```

```
<attribute key="genre" value="Comedy"/>
</qualifier>
</bean>
<bean class="footmark.HelloWorld">
<meta key="format" value="DVD"/>
<meta key="genre" value="Action"/>
</bean>
```

如果@Autowired 注入的是 BeanFactory、ApplicationContext、ResourceLoader 等系统类型，那么则不需要 @Qualifier，此时即使提供了@Qualifier 注解，也将会被忽略；而对于自定义类型的自动装配，如果使用了@Qualifier 注解并且没有名字与之匹配的 Bean，则自动装配匹配失败。

6.6.3.6　使用 JSR-250 中的 @Resource 和@Qualifier 注解

如果希望根据 name 执行自动装配，那么应该使用 JSR-250 提供的@Resource 注解，而不应该使用@Autowired 与@Qualifier 的组合。@Resource 使用 byName 的方式执行自动封装。@Resource 标注可以作用于带一个参数的 Setter 方法、字段，以及带一个参数的普通方法上。@Resource 注解有一个 name 属性，用于指定 Bean 在配置文件中对应的名字。如果没有指定 name 属性，那么默认值就是字段或者属性的名字。@Resource 和@Qualifier 的配合虽然仍然成立，但是@Qualifier 对于@Resource 而言，几乎与 name 属性等效。

6.6.4　Spring3 在构建 RESTful Web Services 的方法

Spring3 后就支持 RESTful Web Services 的开发，在这之前，通常使用 Restlet、RestEasy 和 Jersey 技术来创建 Java 的 RESTful Web Services。目前虽然对 REST 的支持并不是 JAX-RS 的一种实现，但是它具有比标准定义更多的特性；REST 支持被无缝整合到 Spring 的 MVC 层，它可以很容易应用到使用 Spring 构建的应用中。即 Spring REST 支持的主要特性包括：

（1）注释，如@RequestMapping 和@PathVariable，支持资源标识和 URL 映射。

（2）ContentNegotiatingViewResolver 支持为不同的 MIME/内容类型使用不同的表示方式。

（3）使用相似的编程模型无缝地整合到原始的 MVC 层。

通常，首先需要设置 web.xml 文件来激活 Spring WebApplicationContext，然后配置 rest-servlet.xml 文件。其程序片段如下。

配置 web.xml：

```
<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>
    /WEB-INF/rest-context.xml
</param-value>
</context-param>
<!-- This listener will load other application context file in addition to
        rest-servlet.xml -->
```

```
<listener>
<listener class>
    org.springframework.web.context.ContextLoaderListener
</listener-class>
</listener>
<servlet>
<servlet-name>rest</servlet-name>
<servlet-class>
    org.springframework.web.servlet.DispatcherServlet
</servlet-class>
<load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>rest</servlet-name>
<url-pattern>/service/*</url-pattern>
</servlet-mapping>
```

配置 rest-servlet.xml：

```
<context:component-scan base-package="dw.spring3.rest.controller" />
<!--To enable @RequestMapping process on type level and method level-->
<bean class="org.springframework.web.servlet.mvc.annotation
                .DefaultAnnotationHandlerMapping" />
<bean class="org.springframework.web.servlet.mvc.annotation
                .AnnotationMethodHandlerAdapter" />
<!--Use JAXB OXM marshaller to marshall/unmarshall following class-->
<bean id="jaxbMarshaller" class="org.springframework.oxm.jaxb.
Jaxb2Marshaller">
<property name="classesToBeBound">
    <list>
        <value>dw.spring3.rest.bean.*</value>
        <value>dw.spring3.rest.bean.*</value>
    </list>
</property>
</bean>
<bean id=" "  class="org.springframework.web.servlet.view.xml.
MarshallingView">
<constructor-arg ref="jaxbMarshaller" />
</bean>
<bean id="viewResolver" class=
"org.springframework.web.servlet.view.BeanNameViewResolver" />
```

在代码中，Component-scan 启用对带有 Spring 注释的类进行自动扫描，并将检查控制器类中所定义的@Controller 注释。DefaultAnnotationHanlderMappings 和 AnnotationMethod-HandlerAdapter 使用@ReqeustMapping 注释的类或函数的 beans 由 Spring 处理。Jaxb2Mashaller 定义使用 JAXB 2 进行对象 XML 映射（OXM）的编组器（marshaller）和解组器（unmarshaller

MashallingView 定义一个使用 Jaxb2Mashaller 的 XML 表示 viewBeanNameViewResolver 使用用户指定的 bean 名称定义一个视图解析器。

Web Services 的另一个常用特性是它们能够根据请求产生不同的表示。如 HTML/text 方法、application/XML。Spring 3 引入了一个名为 ContentNegotiatingViewResolver 的新视图解析器，它可以根据请求的内容类型（请求头中的 Accept 属性）或 URI 后缀来切换视图解析器。这时，在 rest-servlet.xml 文件中，用注释去掉原来定义的 viewResolver，而使用 ContentNegotiatingViewResolver 来替代它，程序片段如下：

```
<bean class="org.springframework.web.servlet.view
                .ContentNegotiatingViewResolver">
<property name="mediaTypes">
    <map>
    <entry key="xml" value="application/xml"/>
    <entry key="html" value="text/html"/>
    </map>
</property>
<property name="viewResolvers">
    <list>
    <bean class="org.springframework.web.servlet.view
                .BeanNameViewResolver"/>
    <bean class="org.springframework.web.servlet.view.UrlBasedViewResolver">
        <property name="viewClass" value=
        "org.springframework.web.servlet.view.JstlView"/>
        <property name="prefix" value="/WEB-INF/jsp/"/>
        <property name="suffix" value=".jsp"/>
    </bean>
    </list>
</property>
</bean>
```

同时，Spring 也支持对 SOA 的支持，能使 IoC 和 AOP 等方法重构 Web Service 的访问代码，使得业务逻辑与 Web Service 访问解耦，从而提供一个更加灵活和易于扩展的访问模式。因此，在 Spring 框架中，已经为基于 JAX-RPC 的 Web Service 调用提供了一个客户端代理的类工厂实现：JaxRpcPortProxyFactoryBean。在配置文件 bean.xml 中，可以使用 JaxRpcPortProxyFactoryBean 来创建和配置 Web Service。其程序代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
    <bean id="BeanName"
    class="org.springframework.remoting.jaxrpc.JaxRpcPortProxyFactoryBean">
        <property name="serviceInterface">
        <value> </value>
```

```
            </property>
            <property name "wsdlDocumentUrl">
            <value </value>
            </property>
            <property name-"namespaceUri">
                <value>  </value>
            </property>
            <property name="serviceName">
                <value> </value>
            </property>
            <property name="portName">
                <value> </value>
            </property>
            <property name="endpointAddress">
                <value> </value>
            </property>
        </bean>
        <bean id=" "
        class=" ">
            <property name=" ">
                <ref bean=" "/>
            </property>
        </bean>
    </beans>
```

6.7 数据持久化框架

数据持久化是将程序数据在持久状态和瞬时状态间转换的机制，即把数据保存到可永久保存的存储设备中。持久化的主要应用是将内存中的对象存储在关系型的数据库中，当然也可以存储在磁盘文件中、XML 数据文件中。而持久化框架就是以持久化原理建立一种可复用性、操作方便和可靠性健壮的中间件。持久化开源软件框架是将持久化中间件的源代码进行公布，并指出遵循什么样的开源软件协议。下面主要以 Hibernate 为持久化开源软件为主进行叙述，并继续探讨 iBatis 应用方法。

6.7.1 Hibernate

Hibernate 是一种 Java 语言下的对象关系映射解决方案。它是使用 GNU 宽通用公共许可证发行的自由、开源的软件。它为面向对象的领域模型到传统的关系型数据库的映射，提供了一个使用方便的框架。Hibernate 也是目前 Java 开发中最为流行的数据库持久层框架，现已归 JBOSS 所有。Hibernate 不仅负责从 Java 类到数据库表的映射（还包括从 Java 数据类型到 SQL 数据类型的映射），还提供了面向对象的数据查询检索机制，从而极大地缩短的手动处

理 SQL 和 JDBC 上的开发时间。Hibernate 是用于 POJO 的开放源代码持久性框架，它通过 XML 配置文件提供 POJO 到关系数据库表的对象——关系映射。Hibernate 框架是应用程序调用来实现数据持久性的数据访问抽象层。

也就是说 Hibernate 是一个纯 Java 的对象关系映射和持久性框架，它允许用 XML 配置文件把普通 Java 对象映射到关系数据库表。使用 Hibernate 能够节约大量项目开发时间，因为整个 JDBC 层都由这个框架管理。这意味着应用程序的数据访问层位于 Hibernate 之上，完全是从底层数据模型中抽象出来的。比起其他类似的对象关系映射技术（JDO、实体 bean、内部开发等），Hibernate 具有免费的、开源的、已经成熟到良好的程度的，并得到广泛应用等多种优势。为了实现 Hibernate 操作数据，它提供了一个查询语言，称为 Hibernate 查询语言(HQL)，它与 SQL 很相似。如 HQL 编写方法：SELECT * FROM eg.hibernate.mapping.dataobject. Individual WHERE firstName = "John"。Hibernate 也允许用 XML 配置文件把普通 Java 对象映射到关系数据库表；对象和数据的映射使开发人员不需要关心数据库，通过操作数据对象来实现对数据库的操作。要操作，首先要对 hibernate.cfg.xml 进行配置，程序片段结构如下：

```
<!DOCTYPE hibernate-configuration
    PUBLIC "-//Hibernate/Hibernate Configuration DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-x.x.dtd">
 <hibernate-configuration>
    <session-factory>
        <property name="connection.datasource">java:comp/env/jdbc/hibernate
        </property>
        <property name="show sql">false</property>
        <property name="dialect">net.sf.hibernate.dialect.MySQLDialect
        </property>
        <!-- Mapping files -->
    </session-factory>
   ...
   <channel-definition id="my-amf" class="mx.messaging.channels.AMFChannel">
     <endpoint url=
         "http://{server.name}:{server.port}/{context.root}
         /spring/messagebroker/amf"
         class="flex.messaging.endpoints.AMFEndpoint"/>
 </channel-definition>/*建立通道*/
...
<session-factory>
<property name="hibernate.connection.url">
     jdbc:derby:D:/geronimo-1.1/var/derby/SAMPLE;create=true</property>
<property name="hibernate.connection.driver class">
        org.apache.derby.jdbc.EmbeddedDriver</property>
<property name="hibernate.connection.username">APP</property>
<property name-"hibernate.connection.password"></property>
<property name-"dialect">org.hibernate.dialect.DerbyDialect</property>
<property name "myeclipse.connection.profile">MyEclipse Derby</property>
<property name "connection.password">myeclipse</property>
```

```
<property name "connection.driver class">org.apache.derby.jdbc.
EmbeddedDriver</property>
<property name="current session context class">thread</property>
</session-factory>
/*配置数据库，以开源软件 derby 为例，并在 geronimo 下部署*/
 </hibernate-configuration>
```

上面的程序片段包含与数据源有关的信息（数据库 URL、模式名称、用户名、口令等），以及对包含映射信息的其他配置文件的引用等。

Hibernate 是一种新的 ORM 映射工具，它不仅提供了从 Java 类到数据表之间的映射，也提供了数据查询和恢复机制。相对于使用 JDBC 和 SQL 来手工操作数据库，使用 Hibernate，可以大大减少操作数据库的工作量。Hibernate ORM 还提供延迟加载、分布式缓存等高级特性，这样有利于缩短开发周期和降低开发成本。它（如 OpenJPA）实现了 Java Persistence API（JPA）规范，此规范是 Java EE 5 的必备组成部分。

Hibernate 用 XML（*.hbm.xml）文件把 Java 类映射到表，把 Java Beans 属性映射到数据库表。通过 JDBC 支持全部 SQL 数据库管理系统。并且 Hibernate 可与所有流行的 J2EE 应用服务器和 Web 容器完美集成。Hibernate 是按照 GUN 下的 LGPL 许可证发布的开放式源代码应用程序，它是用于 Java 的超高性能的对象/关系持久性和查询服务。而对于 DAO，它封装了访问一个对象或相关对象集中的数据的逻辑。而 Hibernate 中的 DAO 包含 Criteria 查询和 Hibernate Query Language 查询，还有一个 Hibernate SessionFactory。所有面向数据库的逻辑都包含在 DAO 中，这意味着应该通过 DAO 存储和读取普通 Java 对象（POJO）和其他基本数据类型值。DAO 是企业 Java 分层体系结构中一种很典型的模式，通常通过一个服务访问 DAO。

6.7.1.1 Hibernate 基本接口

所有的 Hibernate 类都位于 org.hibernate 包中，这些类及接口就构成了 Hibernate 整个基本结构。即 Hibernate 的核心接口一共有五个，分别为：Session、SessionFactory、Transaction、Query 和 Configuration。这五个核心接口在任何开发中都会用到。通过这些接口，不仅可以对持久化对象进行存取，还能够进行事务控制。

（1）Configuration 接口。cfg.Configuration 指配置会话工厂的引导类，通常用于为 JVM 创建单一会话（或实体管理器）工厂。负责配置并启动 Hibernate，创建 SessionFactory 对象。在 Hibernate 的启动的过程中，Configuration 类的实例首先定位映射文档位置、读取配置，然后创建 SessionFactory 对象。

（2）SessionFactory 接口。SessionFactory 提供 API 以打开 Hibernate 会话，并处理用户请求；通常，每个处理客户机请求的线程都打开一个会话。负责初始化 Hibernate，它充当数据存储源的代理，并负责创建 Session 对象。需要注意的是 SessionFactory 并不是轻量级的，因为一般情况下，一个项目通常只需要一个 SessionFactory 就够，当需要操作多个数据库时，可以为每个数据库指定一个 SessionFactory，如下程序代码就是 Hibernate 运行时配置方法：

```
public class ORMHelper {
```

```
    private static SessionFactory sf;
    protected static synchronized
    SessionFactory getSessionFactory(String name) {
      if (sf == null) {
        sf = new Configuration().configure(name).buildSessionFactory();
      }
      return sf;
    }
    ...
}
```

其实可以通过多种方法在 Hibernate 中配置 SessionFactory。最常见的场景是调用 configure() 方法。如果没有向 configure() 传入名称，它将在类路径的根目录中查找 hibernate.cfg.xml。如果传入 XML 配置文件的名称，它将在类路径上查找该名称。找到 XML 配置文件后，buildSessionFactory() 方法将使用该配置文件中的元数据创建和初始化 SessionFactory。但有些应用程序从 JNDI 注册表查找 SessionFactory，而不使用静态变量，但是在第一次查找时，仍需要调用配置和 buildSessionFactory，因此几乎没有什么效果，而且静态变量是较常用的方法。还可以使用 Configuration#setProperties() 方法以编程方式配置 Hibernate 配置参数，而不使用 configure()方法从文件读取这些参数，这时较好并且频繁使用的方法是外部化 Hibernate 属性。

(3)Session 接口。Session 提供 API 以使在数据库之间存储和加载实体，以及提供 API 以获取事务和创建查询，也负责执行被持久化对象的 CRUD 操作。但需要注意的是 Session 对象是非线程安全的。同时，Hibernate 的 session 不同于 JSP 应用中的 HttpSession。这里当使用 session 这个术语时，其实指的是 Hibernate 中的 session，而以后会将 HttpSession 对象称为用户 session。

通常，应用程序收到客户机请求时，将从 SessionFactory 获取会话，并在请求结束时关闭会话，其中请求可以是 HttpRequest 或对无状态会话 Bean 的调用等。会话提供处理事务和从数据库加载实体(以及将实体存储到数据库)的方法。Hibernate 应用程序通常管理该会话。为了达到目标，它们通常将会话与线程本地存储关联，这样无需将会话作为参数传递到需要访问它的所有方法；相反，它们可以从线程本地存储中检索它。Hibernate 3.0.1 还提供了 etCurrentSession()方法来达到要求，但是通常会找到显式会话管理。如下程序是会话管理片段：

```
public class ORMHelper {
   private static final ThreadLocal tls = new ThreadLocal();
   public static void openSession() {
      Session s = (Session) tls.get();
      if (s == null) {
         s = getSessionFactory("test.cfg.xml").openSession();
         tls.set(s);
      }
   }
```

```
    public static Session getCurrentSession() {
        return (Session) tls.get();
    }
    public static void closeSession() {
        Session s = (Session)tls.get();
        tls.set(null);
        if (s != null && s.isOpen()) s.close();
    }
    ...
}
```

（4）Transaction 接口。Transaction 提供 API 以管理事务，它负责事务相关的操作。它是可选的，开发人员也可以设计编写自己的底层事务处理代码。

Hiberante 是对 JDBC 的轻量级封装。Hiberante 本身并不具备事务管理能力，即 Hibernate 应用程序可以运行于使用不同事务策略的环境中，应用程序也可以运行于使用本地 JDBC 或全局 Java Transaction API（JTA）事务的环境中。使用本地 JDBC 事务是最常见的场景。如果具有异类数据存储（如数据库和消息队列），则 JTA 事务非常有用，它允许将其视为单个事务。

它是将事务的管理委托给底层的 JDBC 或 JTA 来实现事物的管理和调度的。也就是说 Hibernate 支持的事务类型有 JDBC 和 JTA 两种，其中 Hibernate 的 JDBC 事务是基于 JDBCConnection 实现的，它的生命周期是在 Connection 之内的；而 Hibernate 的 JTA 事务类型是由 JTA 容器来管理的，JTA 容器对当前加入事务的众多 Connection 进行调度，实现其事务性要求，JTA 的事务周期可横跨多个 JDBCConnection 生命周期。如下程序代码就是 Hibernate 事务管理片段：

```
public class ORMHelper {
    private static final ThreadLocal tltx = new ThreadLocal();
    public static void beginTransaction() {
        Transaction tx = (Transaction) tltx.get();
        if (tx == null) {
            tx = getCurrentSession().beginTransaction();
            tltx.set(tx);
        }
    }
    public static void commitTransaction() {
        Transaction tx = (Transaction)tltx.get();
        if (tx != null && tx.isActive()) tx.commit();
        tltx.set(null);
    }
    public static void rollbackTransaction() {
        Transaction tx = (Transaction)tltx.get();
        tltx.set(null);
        if (tx != null && tx.isActive()) tx.rollback();
    }
    ...
}
```

（5）Query 接口。Query 提供 API 以执行查询，负责执行各种数据库查询，可以使用 HQL 语言或 SQL 语句两种表达方式。

6.7.1.2　Hibernate 的关系映射

Hibernate 对象关系映射可以在启动时加载的 XML 映射文件进行集中定义。也可以直接使用这些映射文件或从嵌入源代码的 javadoc 样式注释中生成。在 Hibernate 的较新版本中，还可以通过 Java 5 注释（Java5 引入了注释这种类型，它以注释的形式来表达程序中各成员的元数据信息，采用符号@标示）定义对象关系映射。目前，大多数遗留 Hibernate 应用程序使用 XML 映射文件。即先将建立 POJO 类，然后再在编写 Hibernate XML 映射代码：

（1）继承。Hibernate 中可以实现单个表继承和连续继承。对于 Java 基础类包含其所有子类的大多数属性的情况，可以使用单个表映射继承，该表中的一列值标识特定的子类，行所表示的实例就属于此类。如果没有任何列映射到特定的子类，则这些列必须声明为可以为空，因为它们在该子类的数据库行中将为空。将子类与子类中独特的辨别器值一起使用，还要为子类独有的属性定义映射。而单个表继承策略的缺点是如果子类为该实例定义多个非空属性，则非空约束的丢失会带来数据完整性问题。主要优点是，它为类层次范围的实体和查询之间的多态关系提供最佳支持，因为不存在复杂的连接。如下程序代码就是单个表继承映射表示方法：

```
<class name="所定义的 POJO 类名" table="表名" >
  <id name="基础类 ID" column="表列名"/>
  <discriminator column="表列名类型" type="string"/>
  ...
</class>
<subclass name=" " extends=" "  discriminator-value="  ">
  ...
</subclass>
<subclass name=" " extends=" " discriminator-value=" ">
  ...
</subclass>
```

对于基础类不包含所有子类的大多数属性的情况，每个子类将一个包含基础类属性的表与一个单独的连接表一起使用。而对于非继承属性，该表仅包含列。因此，阅读子类实例需要跨基础类表和子类表进行连接。连接继承策略的优点是可以在子类中定义非空属性，而对应的缺点是需要多个连接才能构造实例。它是最灵活的方法，即可以定义新的子类，并将属性添加到现有子类，而无须修改基础类表。例如下面的程序片段：

连接继承 POJO：

```
public class Participant implements Serializable {
    private Long participantId;
    ...
}
public class SalesRep extends Participant {
```

```
    ...
}
public class Administrator extends Participant {
    ...
}
```

连接继承映射表示方法：

```
<class name="Participant" table="T PRTCPNT" >
   <id name="participantId" column="PRTCPNT TID"/>
   ...
</class>
<joined-subclass name="SalesRep" extends="Participant"  table="T SALESREP">
   <key column="PRTCPNT TID"/>
   ...
</joined-subclass>
<joined-subclass name="Administrator" extends="Participant" table="T ADMIN">
   <key column="PRTCPNT TID"/>
   ...
</joined-subclass>
```

（2）关系。对象模型中的对象之间（和数据模型中的表之间）需要多种关系。当数据模型在类似数据类的任何列之间未指定关系时，对象模型必须明确对象之间的关系以支持遍历。此外，数据模型中的关系没有任何固有方向(尽管按一个方向搜索可以比另一个方向更有效)。而对象模型关系固有包含从一个对象到另一个对象的方向。对象模型关系是在数据模型中实现的，其通过一个表中的外键引用另一个表中的主键，具有外键的表称为子对象。其行表示对象，该对象的生命周期依赖于他们引用的对象。因此，其表将包含父对象的外键。由于子对象具有外键，所以它必须始终指向有效的父对象；它不能是孤立的，这就意味着要删除父对象，必须首先删除其子对象或从父对象到子对象执行级联删除。为了维护关系，子对象又称关系的所有者，而父对象称为非所有者。在实现 Hibernate 关系时，Java 程序员必须在两边设置双向关系，但是数据库只需更新一个值来反映这些更改；在子对象（或所有者）中该更新针对外键。因此，在子对象中，对表示外键的属性的更改会传播到数据库，而对父对象中反向属性的更改不会传播到数据库。

一对一关系定义到另一个持久对象的引用，其中子对象的生命周期完全依赖于父对象的生命周期。一对一关系有异常情况。如果发生异常，Hibernate 中的常见做法是将其建模为组件对象，这样子表中的所有属性都将展开到父表中，因此在访问子表的属性时，无须连接父表和子表。例如下面的映射程序片段：

```
<class name=" " table=" ">
    ...
    <id name=" " column=" "/>
    <!--基础类的子类-->
    <component name=" " class=" ">
        ...
```

```
    </component>
</class>
```

多对一关系定义到单个持久对象的引用。尽管多对一关系可以是单向的，但是经常将其定义为一对多双向关系的反向过程。声明多对一关系的实体是子对象（或关系的所有者），因为其表包含外键，而声明多对一关系的实体引用的对象是父对象。由于其表不包含外键，所以它是非所有者或关系的反向。例如下面的映射程序片段：

```
<class name="" table="">
    <many-to-one name="" class="" column="">
    </many-to-one>
    ...
</class>
<!-- 父类 -->
<class name="" table="">
    <id name="" column=""/>
    ...
</class>
```

一对多关系定义到对象集合的引用。由于用例通常需要从父对象到子对象的遍历，而可能需要（也可能不需要）从子对象到父对象的遍历，所以一对多关系是对象模型中最常见的关系类型，这意味着单向一对多关系可以满足大多数情况。在 Hibernate 中，一对多关系的映射通常是通过将列添加到外键的子表完成的，但映射的详细内容是不同的，具体取决于是单向一对多关系，还是双向一对多关系。在单向关系中，子表中的外键列不会映射到子对象中的属性，它在数据模型中，而不是在对象模型中。由于是单向的，所以仅在父对象中有属性，而子对象中没有。此外，必须将外键列定义为可以为空，因为 Hibernate 将首先插入子行（使用 NULL 外键），并在以后更新它。在双向关系中（通常设置 inverse="true"表示双向），对象关系映射比较好，因为子对象中有一个用于外键列的属性，在数据库中该列不必为空。但是，结果对象模型在对象之间有循环依赖关系和更紧密的耦合关系，并需要其他编程来设置关系的两端。例如下面的程序映射代码片段：

```
<class name="" table="">
   <id name="" column=""/>
   <set name="" table="" cascade="all-delete-orphan">
    <!--可以执行 Hibernate 的 all-delete-orphan 功能执行的任务-->
      <key column=""/>
      <one-to-many class="">
   </set>
</class>
<!-- 子类 -->
<class name="" table="">
   ...
</class>
```

多对多关系通过映射表定义到对象集合的引用。在对象模型中，多对多关系并不是全部

通用的，但是它们通常是双向的。与其他双向关系一样，存在所属端和非所属端。在这种情况下，所属端拥有映射表，而不是外键。可以将任意一端指定为所属端，选取哪一端并不重要。例如下面的程序映射片段：

```
<class name="" table="">
   <id name="" column=""/>
   <set name="" table="" inverse="true">
   <!-- inverse="true"表示双向执行-->
      <key column=" "/>
      <many-to-many column=" class="/>
   </set>
</class>
<class name="" table="">
   <id name="" column=""/>
   <set name="" table="">
      <key column=""/>
      <many-to-many column="" class=""/>
   </set>
</class>
```

（3）Hibernate 基本配置说明。在 Hibernate 中，配置 SessionFactory 的最常见方法是在 hibernate.cfg.xml 文件中包括<property>元素，并将该文件放置在类路径的根文件夹中。另一个很少使用的等效方法是在类路径的 hibernate.properties 文件中包括属性。例如下面的 Hibernate 数据库连接配置代码：

```
...
<hibernate-configuration>
  <session-factory>
   <property name="dialect">
      org.hibernate.dialect.DB2Dialect
    </property>
   <property name="connection.driver class">
      com.h2.jcc.DB2Driver
    </property>
   <property name="connection.url">
      jdbc:db2://localhost:50000/HIBTEST
    </property>
   <property name="connection.username">
      db2admin
    </property>
   <property name="connection.password">
      db2admin
    </property>
  </session-factory>
</hibernate configuration>
```

在 Hibernate 中使用基于 XML 的配置方法，则不仅可以指定配置参数，而且可以指定映射文件的位置。这是常见的场景，因为它使能够配置 SessionFactory，而无须以编程方式指定映射文件的位置。例如下面的程序配置代码：

```
...
<hibernate-configuration>
  <session-factory>
    ...
    <!-- Mapping files -->
    <mapping resource="*.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

Hibernate 应用程序的故障排除非常困难，因为 Hibernate 会基于映射文件中指定的元数据生成 SQL 命令。因此，将日志类别配置为查看 Hibernate 提交到数据库的准确 SQL 通常非常有用。如果研究 SQL 不足以诊断问题，Hibernate 中还提供了其他日志类别，但是将发现在类路径的 log4j.properties 文件中配置了日志类别。例如下面的程序代码：

```
...
<hibernate-configuration>
  <session-factory>
    ...
    <property name="show sql">
        true
     </property>
  </session-factory>
</hibernate-configuration>
```

6.7.2　Hibernate 应用方法

6.7.1 小节已从 Hibernate 的接口、映射和基本应用配置三角度进行了概述和基本的应用说明。现以开源软件 Xdoclet[①]为例来简化 Hibernate 应用代码生成为例来进一步叙述 Hibernate 的应用。

6.7.2.1　Xdoclet 简介

XDoclet 是一个开源项目，它通过在 java 源代码中的一些特殊的注释信息，自动生成配置文件、源代码等。目前的版本可以为 Web、EJB、struts、webwork、hibernate、jdo、jmx 等生成描述文件、源码等，Xdoclet 也提供了 ant 的任务 target 支持，完全通过 ant 来完成任务。也就是说 XDoclet 能够很容易成为 Java 编程工具箱中的一个更加通用的跨技术代码生成工具。通过特殊标记的 Java 源文件和预先定义的模板来组合来生成代码，因此，通常具有如

① http://xdoclet.sourceforge.net/xdoclet

下优势：

（1）XDoclet 与 Apache Ant 紧密集成，从而提供了高度自动化的操作。

（2）把控制代码生成和模板处理的 XDoclet 标签作为内联注释嵌入到 Java 源代码文件中，这消除了同步多个相关文件和控制文件的需要。

（3）XDoclet 的内置 Java 解析器使用它对 Java 代码结构的深入理解，为输入的 Java 代码建立内部结构模型，该结构模型又经常被称为元数据（metadata），因为它包含与关联代码有关的数据。

（4）XDoclet 的模板生成逻辑拥有对输入 Java 代码的内部结构模型的完全访问权。图 6-30 所示是 Xdoclet 生成代码基本结构。

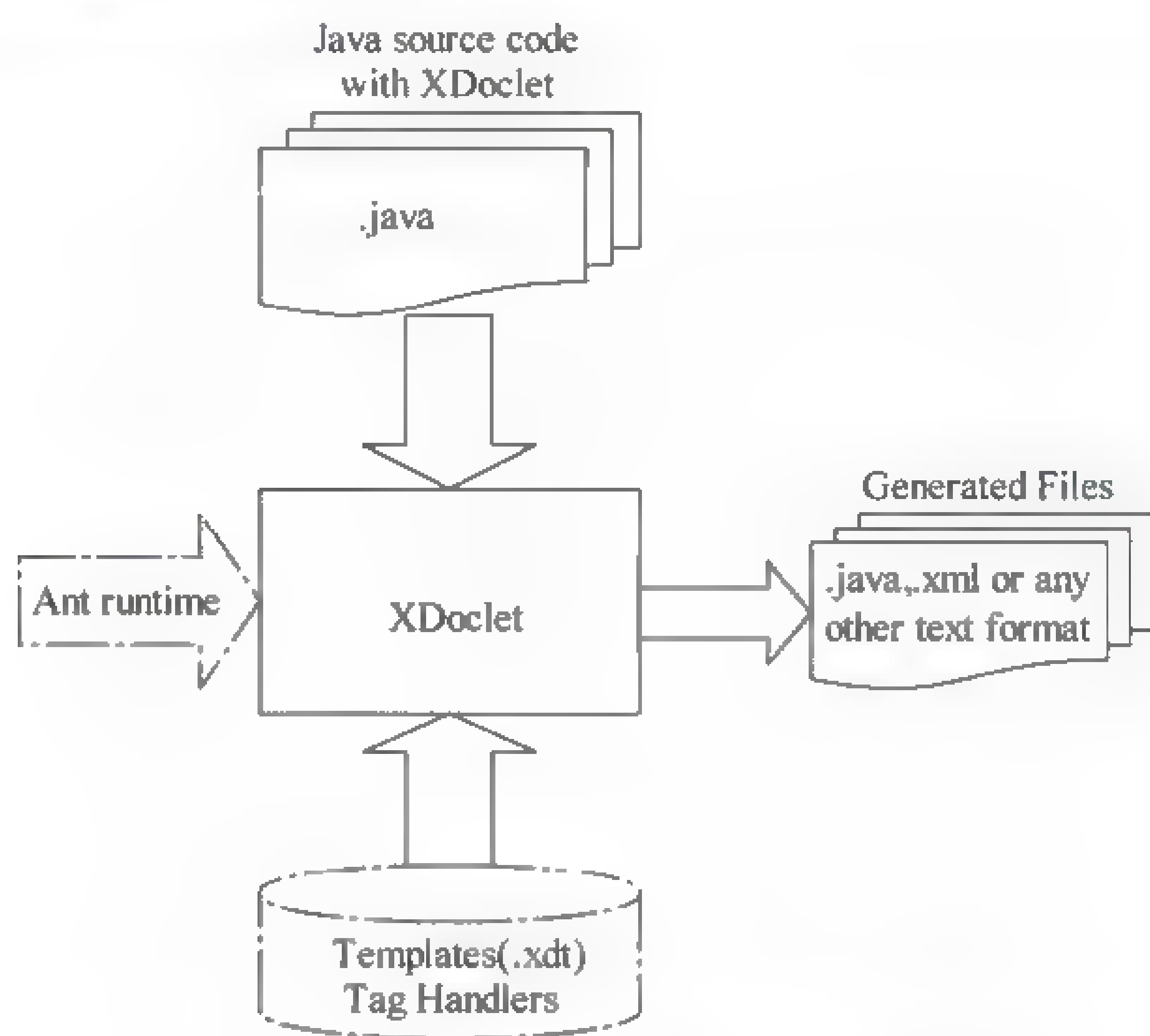

图 6-30 XDoclet 要求的输入和生成的输出

在图 6-30 中，包含嵌入式 XDoclet 标签的 Java 源代码是系统的输入。在 Apache Ant 的驱动下，XDoclet 处理输入的代码，生成的输出文本可以是 Java 源代码、HTML 页面、XML 文件等。同时，为了处理输入，XDoclet 需要使用模板（保存在.xdt 文件中）和标签处理器（用 Java 编码）。

XDoclet 对包含嵌入式 XDoclet 标签的输入 Java 源代码进行解析，并为代码建立非常详细的结构模型。结构模型中的每个元素都代表源代码中的一个 Java 结构。图 6-31 所示的结构模型，揭示了 XDoclet 跟踪的代码构造和关系。

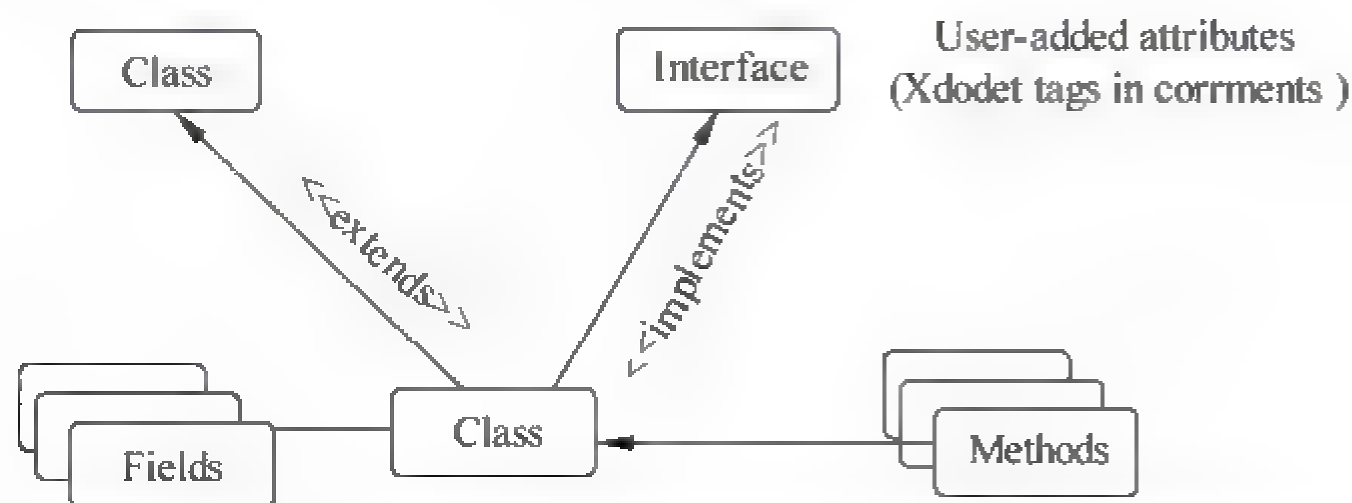

图 6-31 XDoclet 的解析的 Java 源代码的内部结构模型

在图 6-31 中，结构模型跟踪类、接口、方法之类的模型元素，该模型还跟踪元素之间的关系，例如继承和接口实现。以内联注释的形式嵌入在源代码中的 XDoclet 标签被解析为模型元素的属性，并被跟踪。

如图 6-32 所示，Apache Ant 在运行的时候控制着 XDoclet 的配置和操作。XDoclet 解析输入的 Java 源代码，并在内存中生成结构模型。模板引擎通过处理一组模板和标签处理器，生成输出文件。模板和标签处理器可以是内置的，也可以是定制的。在代码生成期间，模板和标签处理器拥有对结构模型的完全访问。

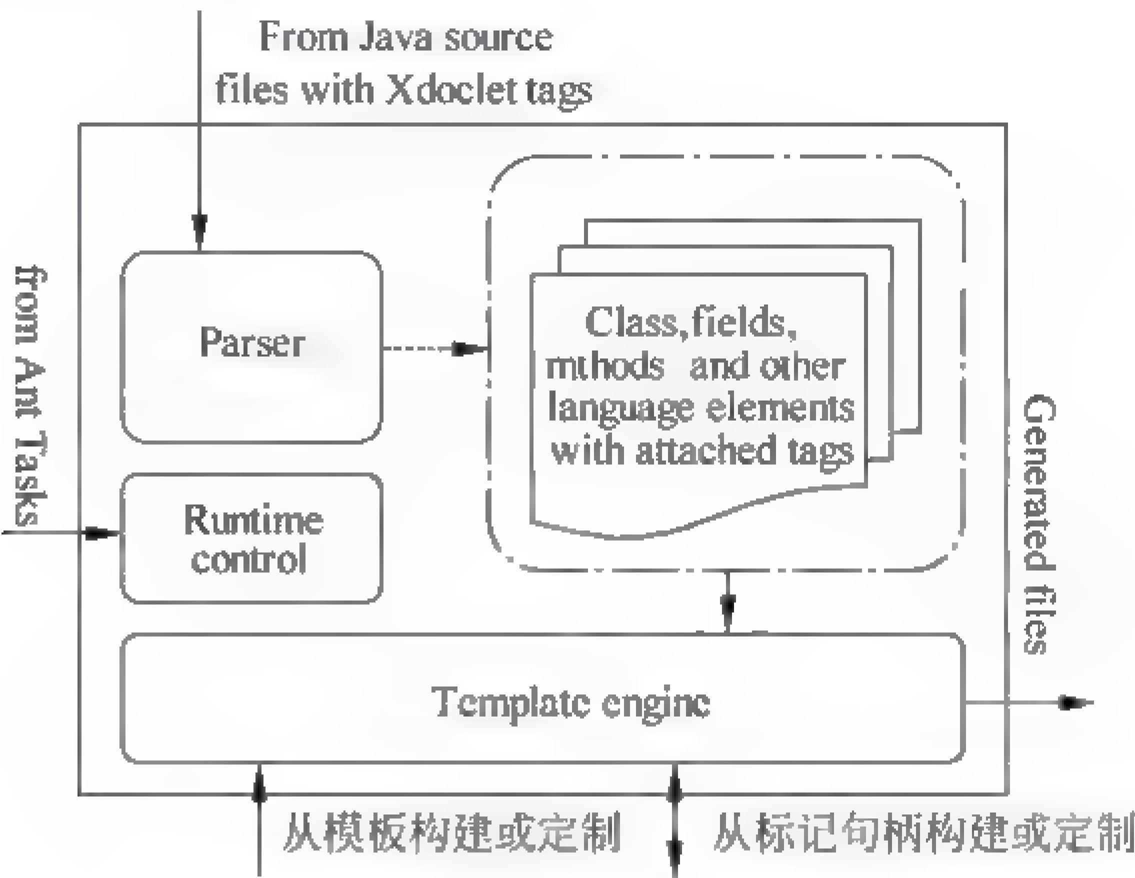

图 6-32　XDoclet 内部的功能块

6.7.2.2　Xdoclet 在 Hibernate 中应用

在发布 XDoclet 的时候，XDoclet 已经可以为 EJB 组件集成、J2EE Web 容器集成、Hibernate 持久性层、Struts 框架、Java 管理扩展（JMX）等生成代码，并通过 Apache Ant，来控制着 XDoclet 的操作。XDoclet 是作为一组 Ant 任务存在的，没有 Ant 则不能执行，如图 6-33 所示。

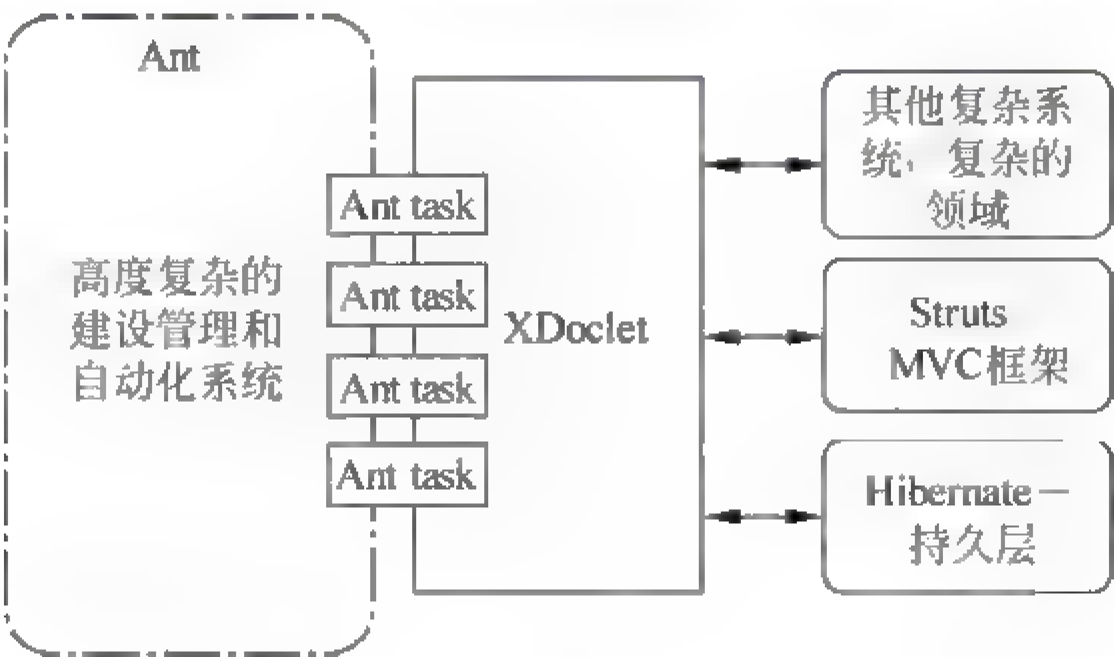

图 6-33　XDoclet 的复杂耦合

如下程序片段就是通过 Xdoclet 来处理 Hibernate 双向一对多关系的代码（是以经典的企业与员工的数据表为例，包括 User、User Group、Roles 和 ContactInfo 类；通常使用 @hibernate.xxx 做为 Xdoclet 的简化指示）：

```
[Group.java]
/**
*
* @return
*
* @hibernate.bag name="users"
* cascade="save-update"
* lazy="true"
* inverse="true"
*
* @hibernate.collection-key
* column="FK_GROUP_ID"
*
* @hibernate.collection-one-to-many
* class="net.sf.hibernateExamples.User"
*/
public List getUsers() {
return users;
}
[User.java]
/**
* @hibernate.many-to-one
* column="FK_GROUP_ID"
* class="net.sf.hibernateExamples.Group"
*/
public Group getGroup() {
return group;
}
```

下面一段程序代码表示在部署到J2EE服务器时，将持久层转换为使用 J2EE 数据源（通过NDI）、JTA 事务和使用 FireBird （一个开放源代码版本的 Interbase）。这是用 Spring 作为 IoC 容器完成的。同时，Spring 允许加入依赖性，在程序片段中的显示了应用程序上下文文件是如何配置 dataSource 的。即描述 dataSource 传递给 sessionFactory，sessionFactory 传递给 UserDAO。

```
<beans>
   <!-- Datasource that works in any application server
      You could easily use J2EE data source instead if this were
      running inside of a J2EE container.
     >
   <bean id-"dataSource" class-"org.apache.commons.dbcp.BasicDataSource">
      <property name-"driverClassName"><value>com.mysql.jdbc.Driver
      </value></property>
      <property name "url"><value>jdbc:mysql://localhost:3306/mysql
      </value></property>
```

```
        <property name "username"><value>root</value></property>
        <property name "password"><value></value></property>
    </bean>
    <!-- Hibernate SessionFactory -->
    <bean id="sessionFactory"
        class="org.springframework.orm.hibernate.LocalSessionFactoryBean">
        <property name="dataSource"><ref local="dataSource"/></property>
                <!-- Must references all OR mapping files. -->
        <property name="mappingResources">
            <list>
                <value>net/sf/hibernateExamples/*.hbm.xml</value>
                <value>net/sf/hibernateExamples/*.hbm.xml</value>
                <value>net/sf/hibernateExamples/*.hbm.xml</value>
                <value>net/sf/hibernateExamples/*.hbm.xml</value>
            </list>
        </property>
        <property name="hibernateProperties">
            <props>
             <prop key="hibernate.dialect">net.sf.hibernate.dialect.
             MySQLDialect</prop>
            </props>
        </property>
    </bean>
   <!-- Pass the session factory to our UserDAO -->
    <bean id="userDAO" class="net.sf.hibernateExamples.UserDAO">
        <property name="sessionFactory"><ref local="sessionFactory"/></property>
    </bean>
</beans>
```

6.7.3 iBatis 应用方法

JpetStore①是 iBatis 的示例程序，是基于 Struts MVC 框架的，以 iBatis 作为持久化层。该示例程序设计优雅，层次清晰，可以学习及作为一个高效率的编程模型参考。iBatis PetStore 应用程序支持在各种应用服务器上执行的多种数据库，且用户界面（UI）框架使用 truts 实现。持久层使用 iBatis SQL Maps 框架和 iBatis 数据访问对象（DAO）框架组成，应用程序此处使用 Derby 数据库进行永久存储，如图 6-34 所示。

iBatis 是一个功能强大实用的 SQL Map 工具，不同于其他 ORM 工具（如 hibernate），它是将 SQL 语句映射成 Java 对象，而对于 ORM 工具，它的 SQL 语句是根据映射定义生成的。iBatis 以 SQL 开发的工作量和数据库移植性上的让步，为系统设计提供了更大的自由空间。有 iBatis 代码生成的工具，可以根据 DDL 自动生成 iBatis 代码，这样能减少很多工作量。对基于 iBatis 的示例程序最初是 Sun 公司的 J2EE petstore，其最主要目的是用于学习 J2EE，

① http://sourceforge.net/projects/ibatisjpetstore/

但是其缺点也很明显，就是过度设计了该程序模型。同时，Oracle 用 J2EE petstore 来比较各应用服务器的性能，微软也推出了基于.Net 平台的 Pet shop，用于竞争 J2EE petstore。而 JpetStore 则是经过改良的基于 Struts 的轻便框架 J2EE web 应用程序，相对来说，JpetStore 设计和架构更优良，各层定义清晰，使用了很多最佳实践和模式，避免了很多反模式，如使用存储过程，在 java 代码中嵌入 SQL 语句，把 HTML 存储在数据库中等等。图 6-35 是面向 JpetStore 应用的构架图。

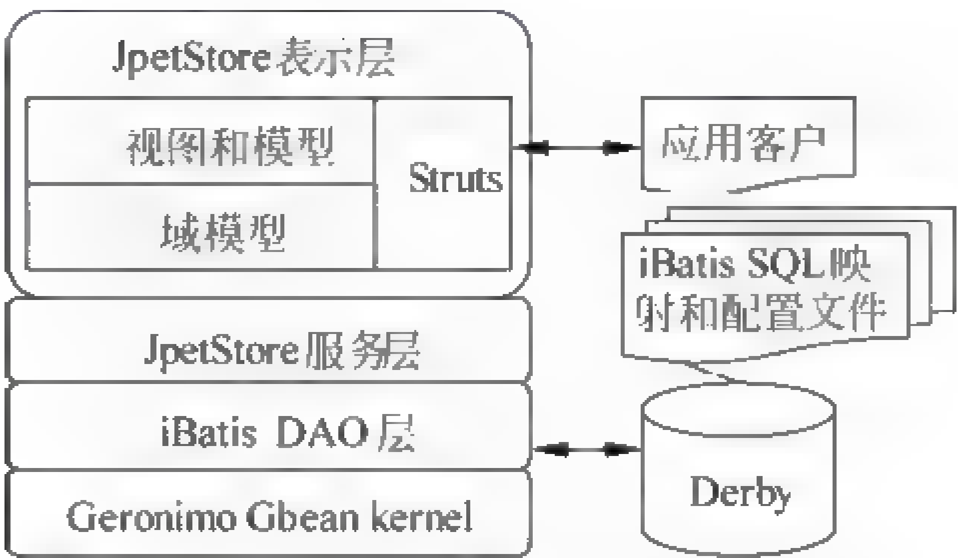

图 6-34 JPetStore 应用程序的组件和框架

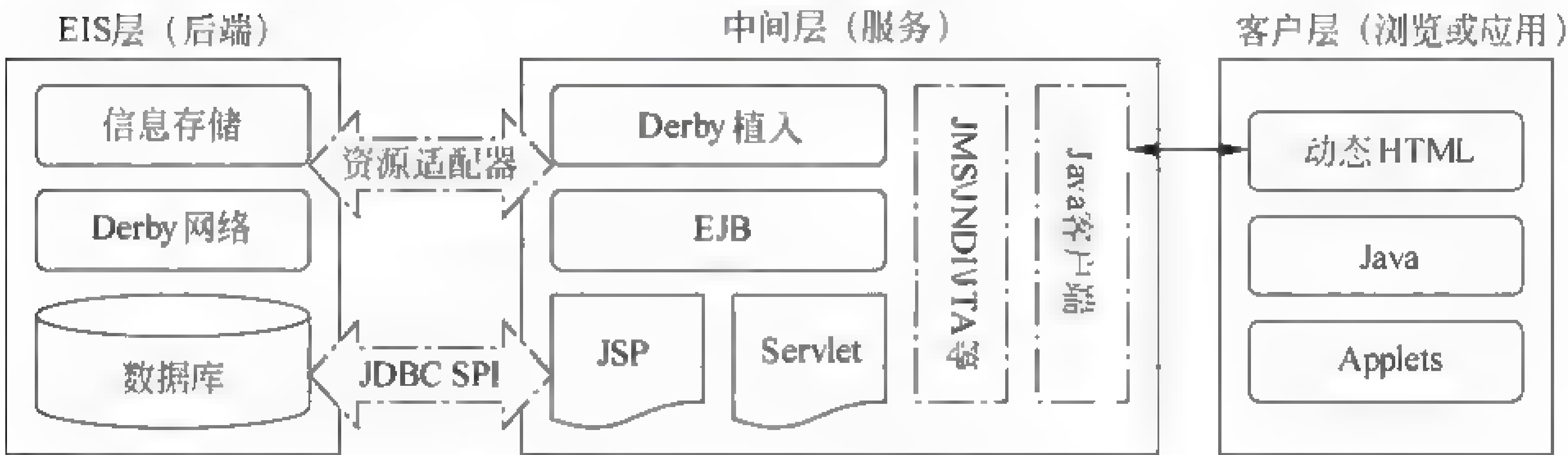

图 6-35 面向 JpetStore 的三层构架结构

下面对 JpetStore 程序结构进行简要叙述，其结构如图 6-36 所示。

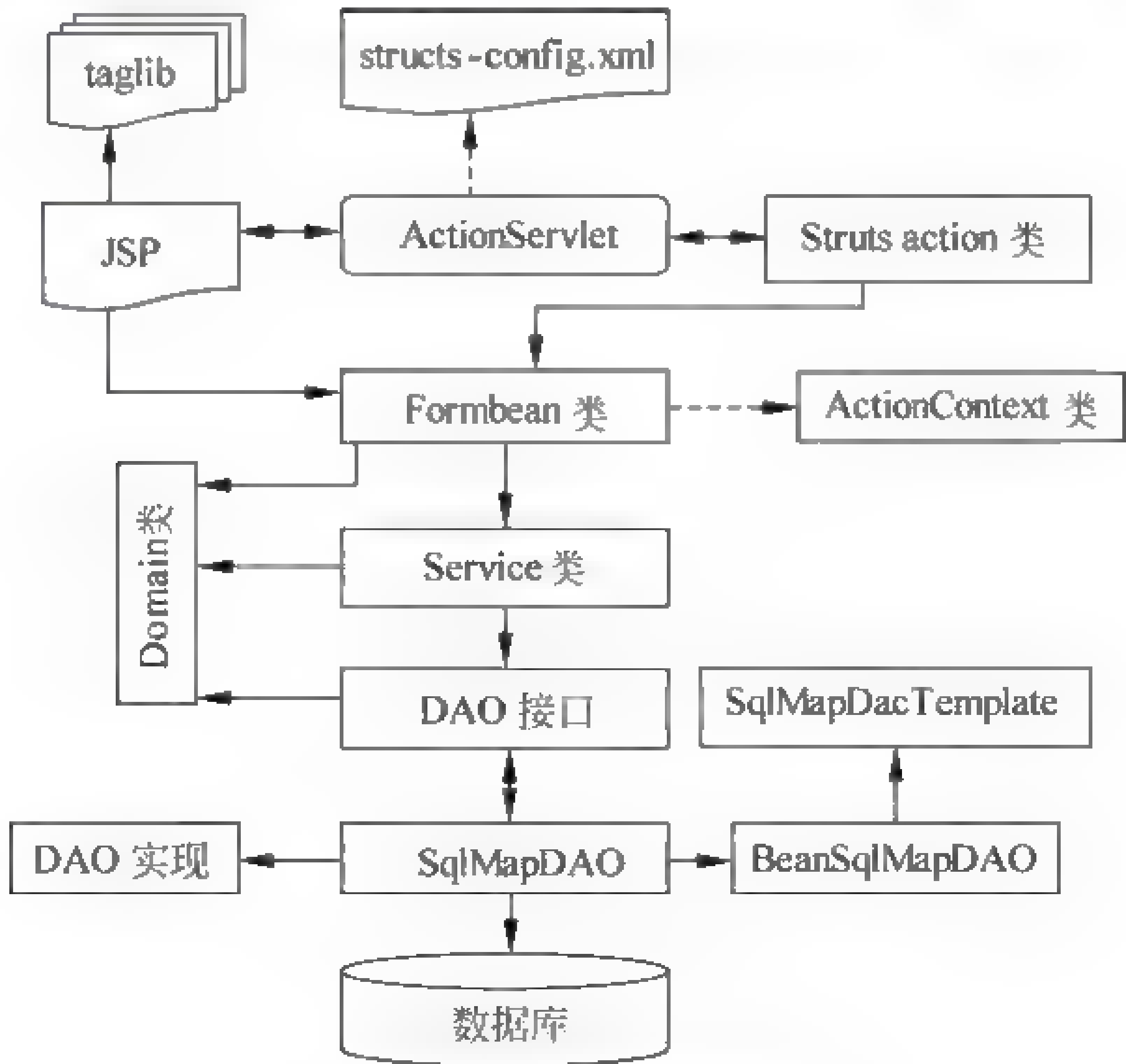

图 6-36 JpetStore 实现

在图 6-36 中：

（1）BeanAction.java：它是唯一一个 Struts action 类，位于 com.ibatis.struts 包下。它也是一个通用的控制类，利用反射机制，把控制转移到 form bean 的某个方法来处理。

（2）Formbean 类：它位于 com.ibatis.jpetstore.presentation 包下。Formbean 类全部继承于 BaseBean 类，而 BaseBean 类实际继承于 ActionForm。因此，Formbean 类就是 Struts 的 ActionForm，Formbean 类的属性数据就由 Struts 框架自动填充。当前的版本进一步扩展了 struts 中 ActionForm 的应用：Form bean 类还具有行为，更像一个业务对象，其行为（方法）由 BeanAction 根据配置（struts-config.xml）的 URL 来调用。在 JpetStore 中，Struts-config.xml 的配置里有三种映射方式，来告诉 BeanAction 把控制转到哪个 form bean 对象的哪个方法来处理。

① URL Pattern

```
 <action path="/shop/viewOrder" type="com.ibatis.struts.BeanAction"
   name="orderBean" scope="session"
   validate="false">
   <forward name="success" path="/order/ViewOrder.jsp"/>
</action>
//控制将被转发到“orderBean”，然后使 form bean 对象的“viewOrder”方法（行为）来处理。方法名取 path 参数的以"/"分隔的最后一部分
```

② Method Parameter

```
 <action path="/shop/viewOrder" type="com.ibatis.struts.BeanAction"
   name="orderBean" parameter="viewOrder" scope="session"
   validate="false">
   <forward name="success" path="/order/ViewOrder.jsp"/>
</action>
//制将被转发到“orderBean”，然后使 form bean 对象的“viewOrder”方法（行为）来处理。配置中的 parameter 参数表示 form bean 类上的方法，parameter 参数优先于"path"参数
```

③ No Method call

```
 <action path="/shop/viewOrder" type="com.ibatis.struts.BeanAction"
   name="orderBean" parameter="*" scope="session"
   validate="false">
   <forward name="success" path="/order/ViewOrder.jsp"/>
  </action>
//form bean 上没有任何方法被调用。如果存在 name 属性，则 Struts 把表单参数等数据填充到 form bean 对象后，把控制转发到 success。否则，如果 name 为空，则直接转发控制到 success
```

（3）Service 类：它位于 com.ibatis.jpetstore.service 包下，属于业务层。这些类封装了业务以及相应的事务控制，Service 类由 form bean 类来调用。

（4）com.ibatis.jpetstore.persistence.iface 包下的类是 DAO 接口，属于业务层，其屏蔽了底层的数据库操作，供具体的 Service 类来调用。DaoConfig 类是工具类（DAO 工厂类），使得 Service 类通过 DaoConfig 类来获得相应的 DAO 接口，而不用关心底层的具体数据库操作。

（5）com.ibatis.jpetstore.persistence.sqlmapdao 包下的类是对应 DAO 接口的具体实现。在 JpetStore 中采用了 iBatis 来实现 ORM。这些实现类继承 BaseSqlMapDao 类，而 BaseSqlMapDao 类则继承 iBatis DAO 框架中的 SqlMapDaoTemplate 类。ibatis 的配置文件存放在 com.ibatis.jpetstore.persistence.sqlmapdao.sql 目录下。这些类和配置文件位于数据层。

（6）Domain 类位于 com.ibatis.jpetstore.domain 包下，是普通的 javabean。在这里用作数据传输对象（DTO），贯穿视图层、业务层和数据层，用于在不同层之间传输数据。

下面继续介绍 JpetStore 应用的配置方法：

（1）如下配置代码是 PetStore 应用程序部署 Derby 数据库的配置方法，该配置是以 Geronimo 做 Web 服务部署软件的。

```
<?xml version="1.0" encoding="UTF-8"?>
<connector xmlns="http://geronimo.apache.org/xml/ns/j2ee/connector-1.1">
  <dep:environment
    xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.1">
    <dep:moduleId>
      <dep:groupId>console.dbpool</dep:groupId>
      <dep:artifactId>JPetstoreDerbyDataSource</dep:artifactId>
      <dep:version>1.0</dep:version>
      <dep:type>rar</dep:type>
    </dep:moduleId>
    <dep:dependencies>
      <dep:dependency>
        <dep:groupId>org.apache.derby</dep:groupId>
        <dep:artifactId>derby</dep:artifactId>
        <dep:version>10.1.1.0</dep:version>
        <dep:type>jar</dep:type>
      </dep:dependency>
    </dep:dependencies>
  </dep:environment>
  <resourceadapter>
    <outbound-resourceadapter>
      <connection-definition>
        <connectionfactory-interface>
          javax.sql.DataSource
        </connectionfactory-interface>
        <connectiondefinition-instance>
          <name>JPetstoreDerbyDataSource</name>
          <config-property-setting name="Password">
            APPdbpw
          </config-property-setting>
          <config-property-setting name="Driver">
```

```
                org.apache.derby.jdbc.EmbeddedDriver
            </config property setting>
            <config property setting name="UserName">
                APP
            </config-property-setting>
            <config-property-setting name="ConnectionURL">
                jdbc:derby:JPetstoreDB
            </config-property-setting>
            <connectionmanager>
                <local-transaction/>
                <single-pool>
                    <max-size>10</max-size>
                    <min-size>0</min-size>
                    <blocking-timeout-milliseconds>
                        5000
                    </blocking-timeout-milliseconds>
                    <idle-timeout-minutes>30</idle-timeout-minutes>
                    <match-one/>
                </single-pool>
            </connectionmanager>
        </connectiondefinition-instance>
      </connection-definition>
    </outbound-resourceadapter>
  </resourceadapter>
</connector>
```

（2）所有的 DAO 实现类还是继承于 BaseSqlMapDao 类，实现相应的 DAO 接口。下面的代码就是 DAO 在 Spring 中的配置（applicationContext.xml）方法：

```
<bean id="dataSource"
      class="org.springframework.jdbc.datasource.DriverManager-
      DataSource">
      <property name="driverClassName">
         <value>org.hsqldb.jdbcDriver</value>
      </property>
      <property name="url">
         <value>jdbc:hsqldb:hsql://localhost/xdb</value>
      </property>
      <property name="username">
         <value>sa</value>
      </property>
      <property name="password">
         <value></value>
      </property>
   </bean>
   <!-- ibatis sqlMapClient config -->
```

```
<bean id="sqlMapClient"
    class="org.springframework.orm.ibatis.SqlMapClientFactoryBean">
    <property name="configLocation">
        <value>
            classpath:com\ibatis\jpetstore\persistence\sqlmapdao\
            sql\sql-map-config.xml
        </value>
    </property>
    <property name="dataSource">
        <ref bean="dataSource"/>
    </property>
</bean>
<!-- Transactions -->
<bean id="TransactionManager"
    class="org.springframework.jdbc.datasource.
    DataSourceTransactionManager">
    <property name="dataSource">
        <ref bean="dataSource"/>
    </property>
</bean>
<!-- persistence layer -->
<bean id="AccountDao"
    class="com.ibatis.jpetstore.persistence.sqlmapdao.
    AccountSqlMapDao">
    <property name="sqlMapClient">
        <ref local="sqlMapClient"/>
    </property>
</bean>
```

6.8 A2JT

仍以开源软件为基础，聚集当前流行、可用性好、可靠性高、语义化、面向服务化的开源软件组成一种具有语义 Web 服务开发框架的系统，命名为 A2JT。同样，这些开源软件也可以与同类型、同领域的其他开源软件集成。

6.8.1 A2JT 介绍

A2JT 是一款集 Web 服务、语义转换（语义推理）的开源框架系统，主要由 Web 服务开源框架 Axis 或 CXF，WSDL 语义转换工具 WSDL2OWL，语义推理开源软件 Jena（其实包含在了 WSDL2OWL 中了），本体编辑工具 Protégé，OMG 的语义 Web 服务编辑工具 WSMO Studio，以及 SOA 开源框架 Tuscany 构成。

6.8.2　Web 服务框架：Axis、CXF

Web 服务是建立可交互操作的分布式应用程序的新平台，它是一套标准，定义了应用程序如何在 Web 上进行交互操作。只要可以通过 Web 服务标准对这些服务进行查询和访问，就可以用任何语言，在任何平台上写 Web 服务。即 Web 服务就是一些模块化的应用程序，这些应用程序能在 Web 上描述、发布、定位和调用。当前，实现 Web 服务的方式最常见的开源框架有两种，一种是 Axis 框架，另一种是 CXF，这两种开源软件都是用于来开发 Web 服务的，下面分别进行叙述其原理和基本应用方法。

6.8.2.1　Axis

Axis 框架来自 Apache 开放源代码组织，它是基于 JAVA 语言的最新的 SOAP 规范（SOAP 1.2）和 SOAP with Attachments 规范（来自 Apache Group）的开放源代码实现。目前有很多流行的开发工具都使用 AXIS 作为其实现支持 Web 服务的功能，例如 JBuilder 及著名的 Eclipse J2EE 插件 Lomboz。

2006 年 5 月推出 Apache Axis2 1.0 是一个大的里程碑。Axis2 1.1 于 2006 年 11 月推出，提供了大量新功能（其中大部分都是其用户最初提出的）及大量错误修补程序（使其更加稳定）。从最初的 Apache Axis 和 Apache SOAP 到目前的 Axis2，经历了很大的发展。它不仅更高效、模块化、基于 XML，而且具有灵活性和可扩展性，实现了安全性和可靠性等企业功能。因此，Axis 2 具有以下新特性：

（1）采用名为 AXIOM（AXIs Object Model，Axis 对象模型）的新核心 XML 处理模型。

（2）支持 In-Only 和 In-Out 消息交换模式（MEP）。

（3）阻塞和非阻塞客户端 API（应用程序编程接口）。

（4）支持内置的 Web 服务寻址（WS-Addressing）。

（5）支持 XMLBeans 数据绑定。

（6）新部署模型。

（7）支持超文本传输协议（HTTP）、简单邮件传输协议（SMTP）和传输控制协议（TCP）等传输协议。

Apache Axis 2 的易用性和功能确实能使其成为了下一代 Web 服务平台。并能很容易地插入到其他相关 Web 服务标准和协议（如 WS-Security、WS-Reliable Messaging 和 WS-Addressing 等）的现实中。图 6-37 所示是 Axis 体系结构。

Axis 2 体系结构将逻辑与状态分离，这允许在并行线程中执行逻辑。服务和调用的静态状态和动态状态分别存储在 Description 和 Context 类中。Axis 2 体系结构是使用七个独立模块实现的。

1．信息模型

此模块管理 SOAP 引擎的状态，它定义一组用于存放状态的类，而引擎管理这些信息对象的生命周期。信息模型包含两种用于存放状态的类：Description 类存放本质上是静态的且存在于 Axis 引擎实例的整个生命周期中的数据（如传输、服务和操作的配置）；Context 类存放调用上下文中有效的服务和操作的动态信息，例如当前请求和响应 SOAP 消息、From 地址、

To 地址和其他元素。

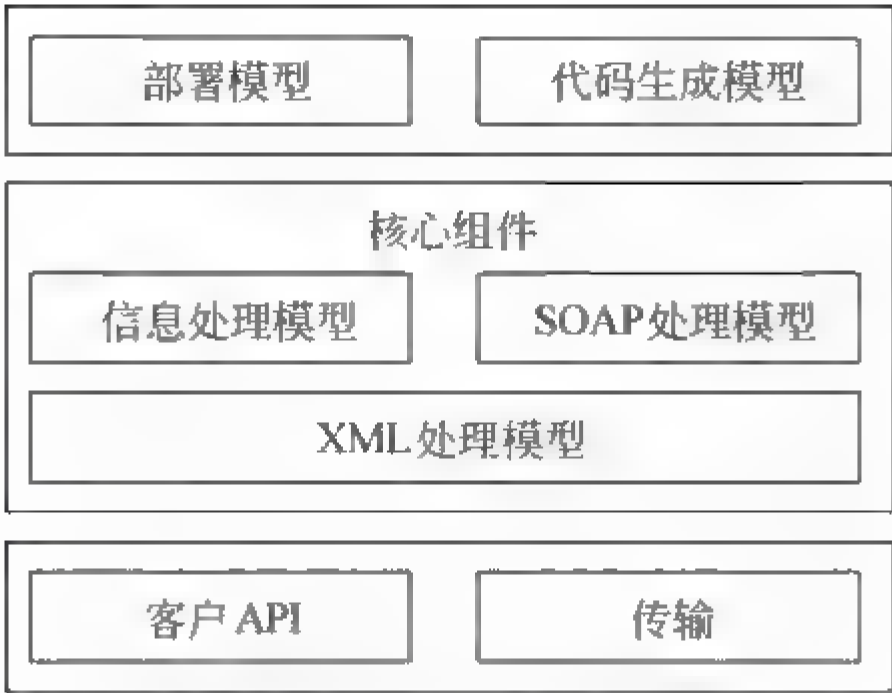

图 6-37　Axis 体系结构

2. XML 处理模型

Axis 2 引入了一个名为 AXIOM 的新模型，用于处理 SOAP 消息。AXIOM 使用 StAX（Streaming API for XML）来解析 XML。StAX 是一个标准的流式 Pull 解析器 Java API。并且 AXIOM 非常精巧，不会减慢 XML 信息集的构建速度。换句话说，对象只有在绝对必要时才会创建。总体而言，AXIOM 和 Axis 2 所占用的内存要小于 Axis 1 所占用的内存。此外，AXIOM 内置了消息传输优化机制（Message Transfer Optimization Mechanism，MTOM）支持。对于 AXIOM 体系结构，可以通过实现 AXIOM 接口并将其插入到 Axis2 中来执行自己的对象模型。同时，由于 AXIOM 最初是作为 Axis 2 的对象模型而开发的，因此 AXIOM 提供了构建于基础 AXIOM API 之上的 SOAP 接口。这允许使用 envelope.getHeaders 和 envelope.getBody 之类的方法查看 SOAP。

Axis 2 对象模型是 Axis 2 的基础，任何 SOAP 消息在 Axis 2 中都表示为 AXIOM。AXIOM 相对于其他 XML 表示形式的优势在于，它基于 pull 解析器技术，而其他大多数则基于 push 解析器技术。pull 与 push 的主要不同之处在于，在 pull 技术中，调用者对解析器具有完全控制权，可以要求下一个事件；而对 push，当要求解析器继续处理时，它将触发事件，直到达到文档最后为止，如图 6-38 所示。

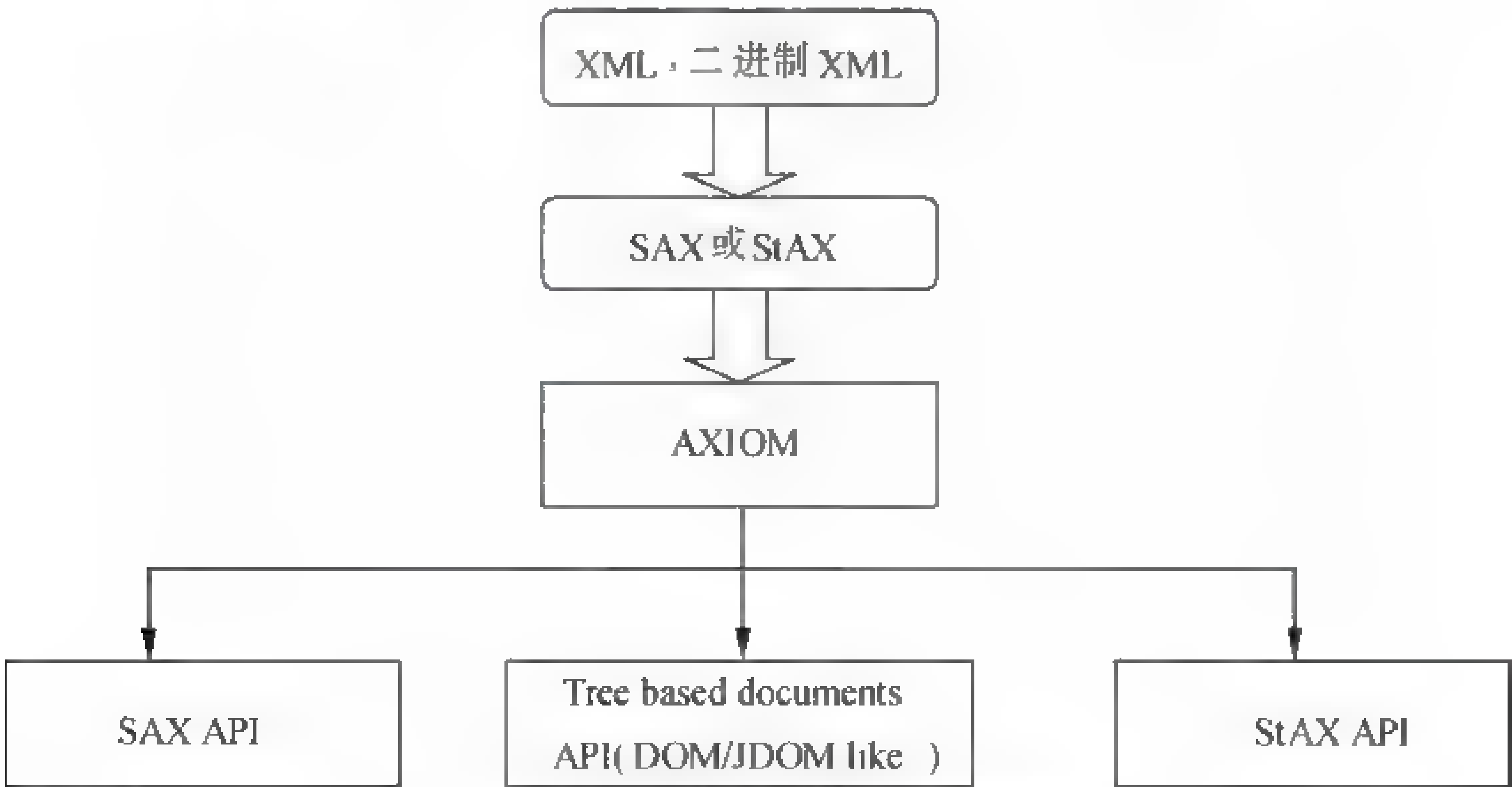

图 6-38　AXIOM 的输入和输出

如下程序代码是创建 AXIOM 方法和从已有代码构造 AXIOM 的方法。

创建 AXIOM 步骤：

（1）建立一个如下 XML 文档

```
<po:line-item po:quantity="2" xmlns:po="http://openuri.org/easypo">
     <po:description>
       Burnham's Celestial Handbook, Vol 2
     </po:description>
       <po:price>19.89</po:price>
</po:line-item>
```

（2）创建 AXIOM 能处理的 XML 读取程序

```
OMFactory factory = OMAbstractFactory.getOMFactory();
//创建内存对象层次结构的第一步是创建一个对象工厂
        OMNamespace poNs =
        factory.createOMNamespace("http://openuri.org/easypo", "po");
        OMElement lineItem =
        factory.createOMElement("line-item", poNs);
        lineItem.addAttribute("quantity", "2", poNs);
        OMElement description =
        factory.createOMElement("description", poNs);
        description.setText("Burnham's Celestial Handbook, Vol 2");
        lineItem.addChild(description);
        OMElement price = factory.createOMElement("price", poNs);
        price.setText("19.89");
        lineItem.addChild(price);
```

（3）使用 StAX writer 来序列化构造好的元素

```
XMLOutputFactory xof = XMLOutputFactory.newInstance();
  XMLStreamWriter writer = xof.
     createXMLStreamWriter(System.out);
     lineItem.serialize(writer);
     writer.flush();
```

从已有代码构造 AXIOM 的方法如下：

通常情况下，Web 服务中的 AXIOM 只需要关心 XML 片段的反序列化。但是在 SOAP 处理中，需要反序列化 SOAP 消息或者经过 MTOM 优化的 MIME 信封。同时，AXIOM 与 SOAP 处理关系特别密切，所以 AXIOM 为此提供内置支持。不过 AXIOM 支持用 SAX 和 StAX 解析器解析 ML。但是，SAX 解析不允许对象模型的延迟构造，因此在延迟构建很重要时，应该使用基于 StAX 的解析器。其步骤如下：

（1）为数据流获得一个 XMLStreamReader

```
File file= new File("line item.xml");
        FileInputStream fis = new FileInputStream(file);
```

```
        XMLInputFactory xif= XMLInputFactory.newInstance();
        XMLStreamReader reader= xif.createXMLStreamReader(fis);
```

（2）创建一个 builder 并将 XMLStreamReader 传递给它

```
StAXOMBuilder builder= new StAXOMBuilder(reader);
        lineItem= builder.getDocumentElement();
```

（3）使用 AXIOM API 来访问属性和子元素或者 XML Infoset 项

```
OMAttribute quantity= lineItem.getFirstAttribute(
        new QName("http://openuri.org/easypo", "quantity"));
   System.out.println("quantity= " + quantity.getValue());
```

（4）访问子元素

```
price= lineItem.getFirstChildWithName(
        new QName("http://openuri.org/easypo", "price"));
              System.out.println("price= " + price.getText());
```

3. SOAP 处理模型

Axis 2 体系结构定义了两个管道(或流)，分别称为 InPipe(InFlow)和 OutPipe(OutFlow)，用于处理服务器端的请求消息和响应消息。在客户端，这两个管道是反向的，换句话说，SOAP 请求消息流经 OutPipe，而响应消息流经 InPipe。管道或流包含一系列分为阶段的处理程序。阶段按照预先定义的顺序执行，如图 6-39 所示。除预先定义的阶段和处理程序集外，用户还可以在操作级别、服务级别或全局级别配置用户阶段和相关处理程序。处理程序充当 SOAP 消息的拦截器，可以处理 SOAP 消息的 Header 或 Body。

图 6-39 Axis2 管道模型

通过组合使用不同数量的 In 和 Ou 管道，Axis2 可以处理任何 MEP（Message Exchange Pattern）。例如，可以使用一个 In 管道和一个 Out 管道来支持进行 In-out 交互。可以使用一个 In 管道来支持 In-Only 交互。这些管道之间的连接由消息接受者进行。并且 Axis 2 可以处理 WSDL 2.0 规范中定义的大部分 MEP，且也可以扩展为支持任何自定义 MEP。最终 InPipe 是通过以下步骤进行配置的：

TransportIn→PreDispatch→Dispatch→PostDispatch→PolicyDetermination→User→phases→Message validation，服务器的 OutPipe 包含以下阶段进行配置：Message initialization→Policy determination→User phases→MessageOut。

同时，每个 Axis 2 管道内部被逻辑划分为阶段（阶段是管道中的处理程序逻辑集）的区域。将按特定的方式对这些阶段进行命名，以表示在该阶段对消息的处理方式。例如，管道中的第一个阶段是 TransportIn 阶段，所有进行传输信息处理的处理程序都可能位于此处。

Dispatch 阶段中的处理程序将标识此消息的目标服务和操作。

调用 In-Only 操作的程序示例代码：

```
try{
  EndpointReference targetEPR = new EndpointReference(
      "http://localhost:8080/axis2/services/StockQuoteService");

  //Make the request message
  OMFactory fac = OMAbstractFactory.getOMFactory();
  OMNamespace omNs = fac.createOMNamespace(
      "http://localhsot:8080/example", "example");
  OMElement payload = fac.createOMElement("subscribe", omNs);
  payload.setText(" ");
    //Send the request
  MessageSender msgSender = new MessageSender();
  msgSender.setTo(targetEPR);
  msgSender.setSenderTransport(Constants.TRANSPORT HTTP);
  msgSender.send("subscribe", payload);
  }catch (AxisFault axisFault) {
      axisFault.printStackTrace();
  }
 //MessageSender.send()发送请求消息并将其立即返回，要使用的传输由 MessageSender.
 setSenderTransport()指定。此示例通过 HTTP 发送消息
```

下面是调用 In-Out 操作之非阻塞单传输模式程序片段示例代码（在此调用模式中，只使用下面的一个传输连接获得非阻塞调用。如果在一个客户端应用程序中要完成多个 Web 服务调用，而且不希望每次调用都阻塞客户端，则需要此类行为。此时，如果响应可用，则调用立即返回且客户端得以回滚）：

```
...
//Create the call
    Call call = new Call();
    call.setTo(targetEPR);
    //Set the transport info.
    call.setTransportInfo(org.apache.axis2.Constants.TRANSPORT HTTP,
      org.apache.axis2.Constants.TRANSPORT HTTP, false);
 //Callback to handle the response
Callback callback = new Callback() {
        public void onComplete(AsyncResult result) {
        System.out.println("Quote = "
          + result.getResponseEnvelope().getBody().getFirstElement()
            .getText());
      }
       public void reportError(Exception e) {
        e.printStackTrace();
```

```
        }
      };
    //Invoke non blocking
      call.invokeNonBlocking("getQuote", payload, callback);
      //Wait till the callback receives the response.
        while (!callback.isComplete()) {
        Thread.sleep(1000);
      }
  call.close();
  ...
```

4．部署模块

此模块配置 Axis 引擎并部署服务和模块。axis2.xml（在 webapps/axis2/WEB-INF 中）包含 Axis 2 引擎的全局配置，还包括全局模块（Global modules）、全局接收器（Global receivers）、传输（Transports）、用户阶段定义（User phase definitions）。下面是针对服务端的程序示例代码：

```
<service name="AxisService" scope="application">
    <description>AxisService</description>
    <messageReceivers>
        <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-only"
 class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"/>
 <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-out"
 class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
    </messageReceivers>
 <parameter name="ServiceClass"> org.test.axis.service.AxisService
 </parameter>
<operation name="getQuote">
    <messageReceiver
      class="org.apache.axis2.receivers.RawXMLINOutMessageReceiver"/>
  </operation>
  <operation name="subscribe">
    <messageReceiver
      class="org.apache.axis2.receivers.RawXMLINOnlyMessageReceiver"/>
  </operation>
 </service>
```

在程序片段中每个<operation>元素定义服务中一个操作的配置，且<operation>的 name 属性应设置为服务实现类中方法的名称。而 messageReceiver 元素定义用于处理此操作的消息接收器。Axis 2 针对 In-Only 和 In-Out 操作提供了两个无数据绑定的内置 MessageReceivers。其中，org.apache.axis2.receivers.RawXMLINOnlyMessageReceiver 用于 In-Only 操作，而 org.apache.axis2.receivers.RawXMLINOutMessageReceiver 用于 In-Out 操作。如果没有指定 messageReceiver，则 Axis 2 将尝试使用 org.apache.axis2.receivers. RawXMLINOutMessageReceiver 作为默认的 messageReceiver。上述 RawXML 消息接收器将传入 SOAP 消息的<Body>的内容作

为 OMElement（OMElement 是 XML 元素的 AXIOM 缩写）传递给服务实现。此操作应作为 OMElement 返回 SOAP 响应的<Body>元素包含的 XML 内容。例如下面的程序片段：

请求：

```
<soap:Body>
     <ser:sayHello>
        <!--Optional:-->
        <ser:args0>Mr Jack</ser:args0>
     </ser:sayHello>
  </soap:Body>
```

响应：

```
<soapenv:Body>
     <ns:sayHelloResponse xmlns:ns="http://service.axis.org">
        <ns:return>Hello Mr Jack</ns:return>
     </ns:sayHelloResponse>
  </soapenv:Body>
```

5．WSDL 和代码生成

此模块从 WSDL 文件中生成客户端存根和服务器框架代码，Axis 2 代码生成器发出采用正确 XML 样式表的 XML 文件，以用所需语言生成代码。下面是 Axis 2 的 WSDL 代码结构：

```
<?xml version="1.0" encoding="UTF-8" ?>
-<wsdl:definitions
 targetNamespace="http://localhost:8080/axis/Hello.jws"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns="http://www.w3.org/2000/xmlns/"
 xmlns:apachesoap="http://xml.apache.org/xml-soap"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:impl="http://localhost:8080/axis/Hello.jws"
 xmlns:intf="http://localhost:8080/axis/Hello.jws"
 xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
 - <wsdl:message name="helloRequest">
 <wsdl:part name="name" type="xsd:string" />
 </wsdl:message>
 + <wsdl:message name="helloResponse">
 - <wsdl:portType name="Hello">
 - <wsdl:operation name="hello" parameterOrder="name">
 <wsdl:input name="helloRequest" message="intf:helloRequest" />
 <wsdl:output name="helloResponse" message="intf:helloResponse" />
 </wsdl:operation>
 </wsdl:portType>
 - <wsdl:binding name "HelloSoapBinding" type "intf:Hello">
```

```
<wsdlsoap:binding style "rpc" transport "http://schemas.xmlsoap.org/
soap/http" />
- <wsdl:operation name="hello">
<wsdlsoap:operation soapAction="" />
- <wsdl:input name="helloRequest">
<wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/
soap/encoding/"
 namespace="http://DefaultNamespace" />
</wsdl:input>- <wsdl:output name="helloResponse">
<wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.
org/soap/encoding/"
 namespace="http://localhost:8080/axis/Hello.jws" />
</wsdl:output>
</wsdl:operation>
</wsdl:binding>
- <wsdl:service name="HelloService">
- <wsdl:port name="Hello" binding="intf:HelloSoapBinding">
<wsdlsoap:address location="http://localhost:8080/axis/Hello.jws" />
</wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

6．客户端 API

Axis2 客户端 API 调用遵循 WSDL 2.0 定义的 In-Only 和 In-Out 消息模式的操作。客户端 API 支持 In-Out 操作的阻塞和非阻塞调用。非阻塞 Web 服务调用是目前 Web 服务中的一个主要需求。同时，还存在一些以非阻塞方式调用 Web 服务的方法。第一个是客户机编程模型，在此模型中，客户机能在不阻塞其应用程序的情况下以非阻塞方式调用服务。第二个方法是传输级非阻塞调用，其中的调用是在两个传输协议之间发生的。可以是 SMTP 之类的两个单向传输协议，也可以为两个 HTTP 之类的双向传输协议。Axis 2 客户机 API 同时支持上述两个非阻塞调用方案。如下程序片段就是非阻塞双向传输模式：

```
    try {
   ...
     Call call = new Call();
    call.setTo(targetEPR);
    call.setTransportInfo(
      Constants.TRANSPORT HTTP, Constants.TRANSPORT HTTP, true);
  //Callback to handle the response
    Callback callback = new Callback() {
    public void onComplete(AsyncResult result) {
      System.out.println("Quote = "
          + result.getResponseEnvelope().getBody().getFirstElement().
        getText());
      }
    public void reportError(Exception e) {
```

```
            e.printStackTrace();
          }
        };
       //Non-Blocking Invocation
        call.invokeNonBlocking("getQuote", payload, callback);
      //Wait till the callback receives the response.
         while (!callback.isComplete()) {
           Thread.sleep(1000);
        }
        call.close();
}catch (AxisFault axisFault) {
      axisFault.printStackTrace();
}catch (Exception ex) {
      ex.printStackTrace();
}
```

Axis 2 引入了用于调用服务的非常方便的客户机 API，此 API 包含两个类，分别名为 ServiceClient 和 OperationClient。ServiceClient API 专门用于只需要发送和接收 XML 的普通用户，而 OperationClient 旨在供希望处理 SOAP Header 和其他一些高级任务的高级用户使用。通过使用 ServiceClient，只能访问 SOAP 主体或有效负载。当然，可以添加 SOAP Header，但无法从服务客户机检索 SOAP Header。为了实现此操作，可以将需要使用 OperationClient，它具有以下用于调用服务的 API：

（1）sendRobust：此 API 的思路是将 XML 块发送给 Web 服务，而不考虑其响应。不过，如果出现了错误，需要知道此情况。因此，此 API 用于调用并不返回值但可能引发异常的服务。

（2）fireAndForget：此 API 只用于发送 XML 块，但并不考虑响应或异常，因此这是调用仅传入的 MEP。

（3）sendReceive：调用具有返回值的服务。这是最常用的 API 之一，可以用于调用传入-传出的 MEP。

（4）sendReceiveNonBlocking：以非阻塞方式调用服务。服务具有返回值时，可以使用此方法。为了使用此方法，必须传递一个回调对象，将在调用完成后立即调用回调对象。

7. 传输

此模块包含与传输层交互的处理程序。传输处理程序有两种类型：TransportListener 和 TransportSender。TransportListener 从传输层接收 SOAP 消息，然后将其传送到 InPipe 进行处理。TransportSender 通过发送指定传输从 OutPipe 接收到的 SOAP 消息。Axis 2 提供 HTTP、SMTP 和 TCP 的处理程序。对于 HTTP 传输，服务器端上的 AxisServlet 和客户端上的一个简单的独立 HTTP 服务器（由 Axis 2 提供）充当 TransportReceiver。

6.8.2.2　CXF

虽然 Axis2 支持了 WSDL 2.0，支持了 HTTP、SMTP、TCP 和 JMS，Axis 2 内置了 Spring 服务支持，完全支持了 WS-Policy；也随附了 WSDL2Java 和 Java2WSDL 工具，同时提供了

与 Axis 2 协同工作的 Eclipse 插件，通过在 Web 容器中部署管理控制台，可以更方便地对 Axis 2 引擎进行远程管理和简化工作。但对 Spring 框架支持力度、简化工作不如 CXF 操作方便。Apache CXF = Celtix + XFire，Apache CXF 的前身叫 Apache CeltiXfire，现在已经正式更名为 Apache CXF 了，CXF 继承了 Celtix 和 XFire 两大开源项目的精华，提供了对 JAX-WS 全面的支持，并且提供了多种 Binding、DataBinding、Transport 以及各种 Format 的支持，可以根据实际项目的需要，采用代码优先（Code First）或者 WSDL 优先（WSDL First）来轻松地实现 Web Services 的发布和使用。因此具有高性能、可扩展、简单且容易使用等优势。

1．CXF 介绍

ApacheCXF 是一个开源的 Services 框架，像 JAX-WS 一样利用 Frontend 编程 API 来构建和开发 Services。这些 Services 可以支持 SOAP、XML/HTTP、RESTful HTTP 或者 CORBA 等多种协议，并且可以在 HTTP、JMS 或者 JBI 等多种传输协议上运行，CXF 大大简化了 Services 的创建，同时它继承了 XFire 传统，一样可以很好地和 Spring 进行无缝集成。即 CXF 具有以下特性：

（1）支持 Web 服务基本标准及其他多种标准。CXF 支持多种 Web 服务标准，包含 SOAP、Basic Profile、WS-Addressing、WS-Policy、WS-ReliableMessaging 和 WS-Security。

支持 JAX-WS、JAX-WSA、JSR-181 和 SAAJ；支持 SOAP 1.1、1.2、WS-I BasicProfile、WS-Security、WS-Addressing、WS-RM 和 WS-Policy；支持 WSDL 1.1、2.0；支持 MTOM 等多种标准。

（2）Frontends。CXF 支持多种 Frontend 编程模型，且实现了 JAX-WS API（遵循 JAX-WS 2.0 TCK 版本）。它也包含一个 simple frontend 允许客户端和 EndPoint 的创建，而不需要 Annotation 注解。CXF 既支持 WSDL 优先开发，也支持从 Java 的代码优先开发模式。

（3）容易使用。CXF 设计得更加直观与容易使用，即有大量简单的 API 用来快速地构建代码优先的 Services，各种 Maven 的插件也使集成更加容易，支持 JAX-WS API，支持 Spring 2.0 更加简化的 XML 配置方式等。

（4）支持二进制和遗留协议。CXF 的设计是一种可插拔的架构，既可以支持 XML，也可以支持非 XML 的类型绑定，比如：JSON 和 CORBA。

（5）支持多种传输方式。这些传输方式包括 Bindings：SOAP、REST/HTTP；Data Bndings：目前支持 JAXB 2.0、Aegis 两种，默认是 JAXB 2.0。XMLBeans、Castor 和 JiBX 数据绑定方式将在 CXF 2.1 版本中得到支持；格式（Format）：XML、JSON；传输方式：HTTP、Servlet、JMS 和 Jabber；可扩展的 API 允许为 CXF 增加其他的 Bindings，以能够支持其他的消息格式，比如：CSV 和固定记录长度。

（6）灵活部署。CXF 灵活部署主要包括轻量级容器：可在 Tomcat 或基于 Spring 的容器中部署 Services；集成 JBI：可以在如 ServiceMix、OpenESB or Petals 等等的 JBI 容器中将它部署为一个服务引擎；集成 SCA：可以部署在如 Tuscany 之类的 SCA 容器中；集成 J2EE：可以在 J2EE 应用服务器中部署 Services，比如 Geronimo、JOnAS、JBoss、WebSphere Application Server、WebLogic Application Server，以及 Jetty 和 Tomcat；独立的 Java 客户端/服务器。

（7）支持多种编程语言。全面支持 JAX-WS 2.0 客户端/服务器编程模型；支持 JAX-WS 2.0 synchronous、asynchronous 和 one-way API's；支持 JAX-WS 2.0 Dynamic Invocation Interface (DII) API；

支持 wrapped and non-wrapped 风格；支持 XML messaging API；支持 JavaScript 和 ECMAScript 4 XML (E4X)，客户端与服务端均支持；通过 Yoko 支持 CORBA；通过 Tuscany 支持 SCA。

（8）代码生成。全面支持 Java to WSDL、WSDL to Java、XSD to WSDL、WSDL to XML、WSDL to SOAP、WSDL to Service，以及用通过 ServiceMix 支持 JBI 代码生成。

2．CXF 与 Axis 2

Apache CXF Web 服务堆栈是来自 Apache Software Foundation 的另一替代选择，Axis 2 堆栈也来自同一组织。尽管它们来自同一组织，Axis 2 和 CXF 就如何配置和交付 Web 服务采用完全不同的方法。

在用户界面方面，CXF 与 Axis 2 的服务堆栈有很多共同之处。两个堆栈都允许从已有 Java 代码开始构建 Web 服务，或从 WSDL Web 服务描述开始，生成使用或实现服务的 Java 代码。而且与其他堆栈一样，CXF 将服务操作建模为方法调用，而将服务端口类型建模为接口。与 Axis 2 类似，CXF 允许选择不同的数据绑定技术。CXF 对 JAXB 2.x 的支持而高于 Axis 2，因为它允许从 WSDL 生成代码时使用 JAXB 标准（而 Axis 2 不允许）。CXF 还允许使用其他数据绑定方法，不过对这些方法的支持不像在 Axis 2 中那样成熟。特别是，只有在使用 JAXB 或 XMLBeans 数据绑定时才能使用 CXF 从 WSDL 生成代码。

CXF 使用的首选服务配置技术（或在 CXF 术语中称为前端）是 JAX-WS 2.x 注释，通常附有 XML 配置文件。CXF 中对 JAX-WS 注释的支持，因而与 Axis 2 相比，CXF 更适合使用 JAX-WS。与其他 JAX-WS 实现一样，CXF 需要服务 WSDL 在运行时可用于客户机。

同其他堆栈一样，CXF 使用由可配置组件组成的请求和响应处理流，且它调用组件 intercepters，而非 handlers，不过除此以外的其他组件是等效组件。同时，CXF 完全支持 WS-Security 和其他扩展技术，将其作为基础下载的一部分。CXF JARs 是模块化的，即可以根据正在使用的技术选择 JARs，并使其成为应用程序的一部分（CXF 安装目录中的 /lib/WHICH_JARS 文件会告诉各种常见用例所需的特定 JARs）。该模块化的负面效应是最终会产生应用程序所需的一长列特定 JARs；从有利的一面来说，它允许控制部署的大小。CXF 通常需要为 Web 服务构建一个 WAR 文件，而非潜在地部署多个服务到单个服务安装上（这正是 Axis 2 所用的方法）。CXF 还以 Jetty 服务器的形式提供一个适合生产使用的集成 HTTP 服务器。

3．CXF 基本应用

在 CXF 中，配置客户端时，只需代替 JAX-WS 参考实现 wsimport 工具使用 CXF wsdl2java 工具即可。CXF 打印大量令人不悦的日志细节并输出到控制台，使用 Java 日志记录，因此为避免此输出，需要设置一个系统属性使其指向一个日志属性文件，设置为仅在有 WARNING 或 SEVERE 信息时输出日志。示例应用程序所用的 Ant build.xml 使用 JVM 参数行<jvmarg value="-Djava.util.logging.config.file=${build-dir}/logging. properties"/>完成这个设置。

当使用 Axis 2 时，是通过创建一个包含服务和数据模型类的 JAR 文件来准备用于部署的服务，然后通过将该 JAR 拖放到 Axis 2 服务器安装目录中的 WEB-INF/servicejars 目录中来部署服务。当使用 CXF 时，则需要创建一个包含服务和数据模型类、CXF 库 JARs 以及一对配置文件（其中一个文件在这两个堆栈中名称不同）的 WAR 文件。WEB-INF/web.xml 文件配置真正的 servlet 处理。如下是 web.xml 程序示例程序代码（是一个标准 servlet 配置文件）：

```
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee">
  <display-name>CXFLibrary</display-name>
  <description>CXF Library Service</description>
  <listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener
  </listener-class>
  </listener>
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
      classpath:META-INF/cxf/cxf.xml
      classpath:META-INF/cxf/cxf-extension-soap.xml
      classpath:META-INF/cxf/cxf-servlet.xml
     </param-value>
  </context-param>
  <servlet>
    <servlet-name>CXFServlet</servlet-name>
    <servlet-class>org.apache.cxf.transport.servlet.CXFServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>CXFServlet</servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>
</web-app>
```

通常，一个独立文件 WEB-INF/cxf-servlet.xml 用于配置 CXF，使其将 servlet 接收的请求路由到服务实现代码并按需提供服务 WSDL。如下程序片段是 cxf-servlet.xml 配置示例：

```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:jaxws="http://cxf.apache.org/jaxws"
      xmlns:soap="http://cxf.apache.org/bindings/soap"
      xsi:schemaLocation="
         http://www.springframework.org/schema/beans
         http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
         http://cxf.apache.org/jaxws
         http://cxf.apache.org/schemas/jaxws.xsd">
    <jaxws:endpoint  id="Processor"
       implementor="com.sosnoski.ws.library.cxf.CXFLibraryImpl"
       wsdlLocation="WEB-INF/wsdl/library.wsdl"
       address="/">
    </jaxws:endpoint>
</beans>
```

WEB-INF/cxf-servlet.xml 文件只有一个端点定义，其中包括一个实现类、请求的匹配模式，以及 WSDL 文档位置。WSDL 文档位置是这个端点定义中唯一的可选项。如果在 cxf-servlet.xml 文件中不指定服务端点的 WSDL 文档，CXF 会在运行时基于 JAX-WS 注释自动生成一个 WSDL 文档。

在构建应用程序时，将 cxf-home 属性的值改为 CXF 安装目录的路径。如果要使用不同的系统或端口上的服务器进行测试，那么需要更改 host-name 和 host-port。若要使用所提供的 Ant build.xml 构建示例应用程序，例如下面的 build.xml 示例程序代码：

```
<project name="service" basedir="." default="war">
    <property name="dist.dir" value="${basedir}/WEB-INF" />
    <property name="dist.dir.classes" value="${dist.dir}/classes" />
    <path id="build.class.path">
        <fileset dir="${basedir}/lib">
            <include name="*.jar" />
        </fileset>
    </path>
    <target name="war" depends="">
        <javac srcdir="src" destdir="${dist.dir.classes}"
            includes="org/ spring/service/*/*">
            <classpath refid="build.class.path" />
        </javac>
        <war destfile="${dist.dir}/cxfWithSpring.war"
            webxml="${dist.dir}/web.xml">
            <classes dir="${dist.dir.classes}" />
            <lib dir="${basedir}/lib" />
            <webinf dir="${dist.dir}">
                <include name="beans.xml" />
            </webinf>
        </war>
    </target>
</project>
```

这时，打开一个控制台，进入下载文件的根目录，输入 ant。这将首先调用 CXF wsdl2java 工具（包括在 CXF 中），然后编译客户端和服务器，最后将服务器代码打包为 WAR。接着可以将生成的 cxf-library.war 文件部署到测试服务器，并在控制台输入 ant run 尝试运行示例客户端。示例客户端运行，经过一系列对服务器的请求，打印出每个请求的简要结果。

运行 ant 脚本，然后将 cxfWithSpring.war 放到 Tomcat 的部署目录下，运行 Tomcat，待 server 启动完成后，输入 WSDL 路径，将会看到 WSDL 文件。

在 cxf-servlet.xml 配置文件中 Spring Framework bean 配置的使用。前面已经叙述了 Spring 是一种开源应用程序框架，它包括许多可用来装配应用程序的组件库。IoC 是 Spring Framework 的原始基础，它允许链接和配置 JavaBean 类型的软件组件，在运行时使用 Java 映像访问 bean 对象的属性。Spring IoC 容器通常为依赖性信息使用 XML 文件。cxf-servlet.xml 文件就是这种 Spring 配置的一个示例。<beans>元素仅是单个 bean 配置的一个包装器。

<jaxws:endpoint>元素就是这样的一个 bean，CXF 通过特定类型的对象（一个 org.apache.cxf.jaxws.EndpointImpl 实例）与其相关联。同时。除了 JAX-WS 注释之外，Spring 还用于 CXF 堆栈的所有配置，包括 CXF 内部消息流的组织。大部分时候，这些配置细节都通过使用直接包含在 CXF JARs 中的 XML 配置得到自动处理。也就是 CXF 支持 Spring 2.0 Schema 标签配置方式，并且提供快捷暴露 Web 服务的标签。这时，在使用 CXF 与 Spring 进行配合使用时，首先通过 Spring 与 CXF 的配置来定义 Web 服务的客户端 Bean，在 CXF 安装的 src 目录下创建 beanRefClient.xml 配置文件（见前面所述配置文件）。同样，也需要引入 Spring 与 CXF 命名空间的声明，并引入 CXF 的 Bean 的定义文件，最后通过与服务端配置相对的 CXF 标签<jaxws:client>来定义客户端访问服务的声明。如下程序代码就是 CXF 与 Spring 的配置：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:jaxws="http://cxf.apache.org/jaxws"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
        http://cxf.apache.org/jaxws
        http://cxf.apache.org/schemas/jaxws.xsd">
    <!-- Import Apache CXF Bean Definition -->
    <import resource="classpath:META-INF/cxf/cxf.xml"/>
    <import resource="classpath:META-INF/cxf/cxf-extension-soap.xml"/>
    <import resource="classpath:META-INF/cxf/cxf-servlet.xml"/>
    <!-- SurveyWebService Client -->
    <jaxws:client id="surveyServiceClient"
        serviceClass="ws.cxf.ISurveyService"
        address="http://localhost:8080/CXF Spring Survey/SurveyWebService"/>
</beans>
```

6.8.3 服务功能语义转换：WSDL2OWL-S

WSDL2OWL-S 是 2004 年 5 月发布的一款支持 WSDL1.1 的语义 Web 服务转换框架[①]，并以开源软件 GNU Lesser General Public License（LGPL）协议发布。支持 XML、XML Schema 输入，支持 OWL、RDF 输出，支持接口包括 API、SOAP、Web Interface。有关 WSDL2OWL-S 应用示例将在后面章节进行详细。图 6-40 所示是 WSDL2OWL-S 的图形界面运行结果。

6.8.4 语义推理：Jena

Jena[②]是一种构建语义 Web 应用的 Java 开源框架，它是惠普实验室的开放源代码 Jena

① http://semwebcentral.org/projects/wsdl2owl-s

② http://jena.sourceforge.net

Semantic Web Framework，它为 RDF、RDFS、OWL 和 SPARQL 提供了一个程序框架，并且包含了一个基于规则的接口。Jena 框架包括一个 RDF 接口、一个 OWL 接口、SPARQL 查询引擎、记忆和持久存储，以及在 RDF/XML、N3 和 N-Triples 中读与写 RDF。

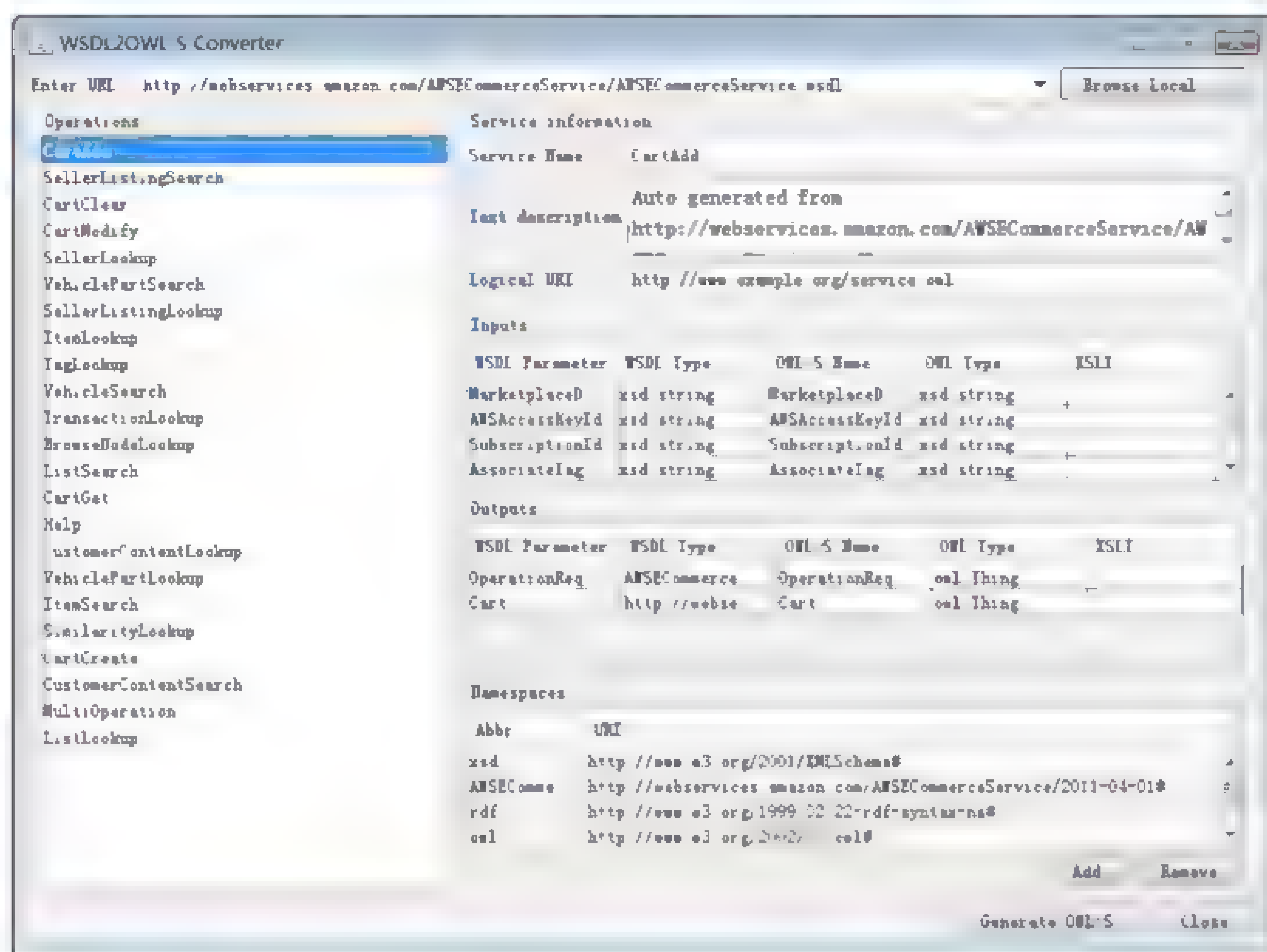

图 6-40 基于 WSDL2OWL-S 的语义 Web 服务结果

6.8.4.1 Jena 简介及基本应用方法

当通过 Jena 的 API 来使用模型时，为模型词汇表中的每个属性定义常量非常有用。如果有词汇表的 RDF、DAML 或 OWL 表示，Jena 的 Schemagen 工具可以自动生成这些常量。Schemagen 在命令行中运行，使用的参数包括模式或本体文件的位置、要输出的类的名称和 Java 包。然后可以导出生成的 Java 类，其 Property 常量用于访问模型。同时，还可以使用 Ant 将 Schemagen 作为构建处理的一部分来运行，保持 Java 常量类与正在变化的词汇表保持同步。

Jena 的 ModelFactory 类是创建不同类型模型的首选方式。在这种情况下，想要空的内存模型，所以要调用的方法是 ModelFactory.createDefaultModel()，这种方法返回 Model 实例。在 Jena 中，语句的主题永远是 Resource，谓词由 Property 表示，对象是另一个 Resource 或常量值，常量在 Jena 中通过 Literal 类型表示，所有这些类型共享公共接口 RDFNode。将需要四个不同的 Property 实例表示所创建树中的关系，这些实例使用 Model.createProperty()创建。将语句添加到模型中的最简单方法是通过调用 Resource.addProperty()，此方法以 Resource 作为主题在模型中创建语句，并且使用两个参数表示语句谓词的 Property 和语句的对象。addProperty()方法被过载：一个过载使用 RDFNode 作为对象，所以可以使用 Resource 或 Literal。还有有益过载，它们使用由 Java 原语或 String 表示的常量。通过使用三元组的主题、谓词和对象调用 Model.createStatement()，还可以直接在模型上创建语句。注意以此种方式创建 Statement 不将其添加到模型中。如果想将其添加到模型中，可以使用创建的 Statement 调

用 Model.add()。同时，将使用来自关系词汇表的属性 siblingOf、spouseOf、parentOf 和 childOf 来描述不同的关系类型。如下程序代码是创建模型的示例：

```
String familyUri = "http://...";
 String relationshipUri = "http://purl.org/vocab/relationship/";
 Model model = ModelFactory.createDefaultModel();
 //创建一个空模型
 URI Resource adam = model.createResource( +" ");
 Resource beth = model.createResource( +" ");
 ...
 Property childOf = model.createProperty(relationshipUri,"childOf");
 Property parentOf = model.createProperty(relationshipUri,"parentOf");
 Property siblingOf = model.createProperty(relationshipUri,"siblingOf");
 Property spouseOf = model.createProperty(relationshipUri,"spouseOf");
 adam.addProperty(siblingOf,beth);
 ...
 Statement statement = model.createStatement(adam,parentOf,fran);
 model.add(statement);
```

不是所有的应用程序都从空模型开始。更常见的是，在开始时从现有数据填充模型。在这种情况下，使用内存模型的缺点是每次启动应用程序时都要从头重新填充模型。另外，每次关闭应用程序时，对内存模型进行的更改都将丢失。为了解决这个问题，一种解决方案是使用 Model.write()序列化模型到文件系统，然后在开始时使用 Model.read()将其取消序列化。不过，Jena 还提供了持久化模型，它们会被持续而透明地持久存储到后备存储器，这使得 Jena 可以在文件系统中或在关系数据库中持久化它的模型。当前支持的数据库引擎是 PostgreSQL、Oracle 和 MySQL。如下代码是导入到 MySQL 持久化模型的完整过程示例：

```
Class.forName("com.mysql.jdbc.Driver");
//初始化 MySQL
DBConnection connection = new DBConnection(DB URL, DB USER, DB PASSWORD,
DB TYPE);
//创建数据库连接
ModelMaker maker = ModelFactory.createModelRDBMaker(connection);
//获得数据
Model testModel = maker.createModel("示例 RDF 数据",true);
model.begin();
InputStream in = this.getClass().getClassLoader().getResource-
AsStream(filename);
model.read(in,null);
commit();
```

在程序中，创建数据库后台模型的第一步是说明 MySQL 驱动类，并创建 DBConnection 实例。DBConnection 构造函数使用用户的 ID 和密码登录到数据库。它还使用包含 Jena 使用的 MySQL 数据库名称的数据库 URL 参数，格式为"jdbc:mysql://localhost/dbname"，Jena 可以在一个数据库内创建多个模型。DBConnection 的最后一个参数是数据库类型。然后

DBConnection 实例可以与 Jena 的 ModelFactory 一起使用来创建数据库后台模型。创建了模型后，可以从文件系统中读入 RDF 文档。不同的 Model.read()方法可以从 Reader、InputStream 或 URL 填充模型。这时可以通过 Notation3、N-Triples 或默认情况下通过 RDF/XML 语法解析模型。

RDQL 是 RDF 的查询语言。虽然 RDQL 还不是正是的标准，但已由 RDF 框架广泛执行，允许简明地表达复杂的查询，查询引擎执行访问数据模型的繁重工作，它的语法表面上类似 SQL 的语法。使用 jena.rdfquery 工具可以在命令行上对 Jena 模型执行 RDQL 查询。使用 RDFQuery 获取 RDQL 查询，然后对指定的模型运行该查询。如下程序片段是从命令行运行 RDQL 查询：

```
$java jena.rdfquery --data jdbc:mysql://localhost/jena --user dbuser
--password dbpass
```

而 Jena 的 com.hp.hpl.jena.rdql 包包含在 Java 代码中使用 RDQL 所需的所有类和接口。要创建 RDQL 查询，将 RDQL 放入 String 中，并将其传送给 Query 的构造函数。通常直接设置模型用作查询的源，除非在 RDQL 中使用 FROM 子句指定了其他的源。一旦创建了 Query，可以从它创建 QueryEngine，然后执行查询。如下示例代码是创建和运行 RDQL 查询：

```
Query query = new Query(queryString);
query.setSource(model);
QueryEngine qe = new QueryEngine(query);
QueryResults results = qe.exec();
```

使用 Query 的一个非常有用的方法是在执行之前将它的一些变量设置为固定值。这种使用模式与 javax.sql.PreparedStatement 的相似。变量通过 ResultBinding 对象与值绑定，执行时该对象会传送给 QueryEngine，并可以将变量与 Jena Resource 或与常量值绑定。在将常量与变量绑定之前，通过调用 Model.createLiteral 将其打包。如下示例代码是将查询变量与值绑定：

```
Query query = new Query(queryString);
ResultBinding initialBinding = new ResultBinding() ;
Resource someResource = getSomeResource();
initialBinding.add("x", someResource);
RDFNode foo = model.createLiteral("bar");
initialBinding.add("y", foo);
QueryEngine qe = new QueryEngine(query);
QueryResults results = qe.exec(initialBinding);
```

QueryEngine.exec()返回的 QueryResults 对象执行 java.util.Iterator，next()方法返回 ResultBinding 对象。查询中使用的所有变量都可以凭名称通过 ResultBinding 获得，而不管它们是否是 SELECT 子句的一部分，查询示例代码如下：

```
SELECT
     ?definition
 WHERE
     (?concept, <wn:查询条件>, "查询词"),
```

```
    (?concept, <wn:查询条件>, ?definition)
USING
    wn FOR <http://...>";
```

concept 变量表示 RDF 资源，所以从 ResultBinding.get()获得的 Object 可以转换成 Resource，然后可以调用 Resource 的查询方法来进一步探查这部分模型，程序代码如下：

```
QueryResults results = qe.exec();
while (results.hasNext()) {
    ResultBinding binding = (ResultBinding)results.next();
    RDFNode definition = (RDFNode) binding.get("definition");
    System.out.println(definition.toString());
    Resource concept = (Resource)binding.get("concept");
    List wordforms = concept.listObjectsOfProperty(...);
}
```

OWL（Web Ontology Language）是 W3C 推荐标准，设计用来明确表示词汇表中词语的意义以及那些词语之间的关系，与 RDF Schema 一起，OWL 提供了一种正式地描述 RDF 模型的机制。除了定义资源可以属于的层次结构类，OWL 还允许表达资源的属性特征。在 Jena 中，本体被看作一种特殊类型的 RDF 模型 OntModel。此接口允许程序化地对本地进行操作，可以使用 OWL 方法创建类、属性限制等。备选方法将本体看作特殊 RDF 模型，仅添加定义其语义规则的语句。如下程序是创建 OWL 本体模型的代码：

```
OntModel wnOntology = ModelFactory.createOntologyModel();
wnOntology.createTransitiveProperty(test.hyponymOf.getURI());
wnOntology.add(test.hyponymOf, RDF.type, OWL.TransitiveProperty);
```

当给定了本体和模型后，Jena 的推理引擎可以派生模型未明确表达的其他语句。Jena 提供了多个 Reasoner 类型来使用不同类型的本体，因此 Jena 提供了 OWLReasoner。并一般从 ReasonerRegistry 中获得 OWLReasoner，ReasonerRegistry.getOWLReasoner()在它的标准配置中返回 OWL reasoner，这对于此简单情况已经足够，此操作返回可以应用本体规则的 reasoner。从原始数据和 OWL 本体创建了推理模型后，它就可以像其他 Model 实例一样进行处理，示例程序片段如下：

```
ModelMaker maker = ModelFactory.createModelRDBMaker(connection);
Model model = maker.openModel(" ",true);
Reasoner owlReasoner = ReasonerRegistry.getOWLReasoner();
Reasoner wnReasoner = owlReasoner.bindSchema(wnOntology);
InfModel infModel = ModelFactory.createInfModel(wnReasoner, model);
query query.setSource(infModel);
QueryEngine qe = new QueryEngine(query);
QueryResults results = qe.exec(initialBinding);
```

6.8.4.2 在 Jena 使用 SPARQL 基本方法

SPARQL 是一种用于 RDF 上的查询语言，它的名字是一个递归缩写，代表“SPARQL

Protocol and RDF Query Language（SPARQL 协议与 RDF 查询语言）”。它的标准化为万维网联盟的 RDF 数据访问工作小组（DAWG）所进行，被认为是语义网科技的一个关键。2008 年 1 月 15 日，SPARQL 正式成为一项 W3C 推荐标准。一般一个 SPARQL 查询由组合、与逻辑、或逻辑，及选项组合所组成。SPARQL 对于语义 Web 就像 SQL 对于关系数据库一样重要。它允许应用程序对分布式 RDF 数据库进行复杂的查询，并得到了互相竞争的多种框架的支持。图 6-41 是 SPARQL 应用模式结构图。

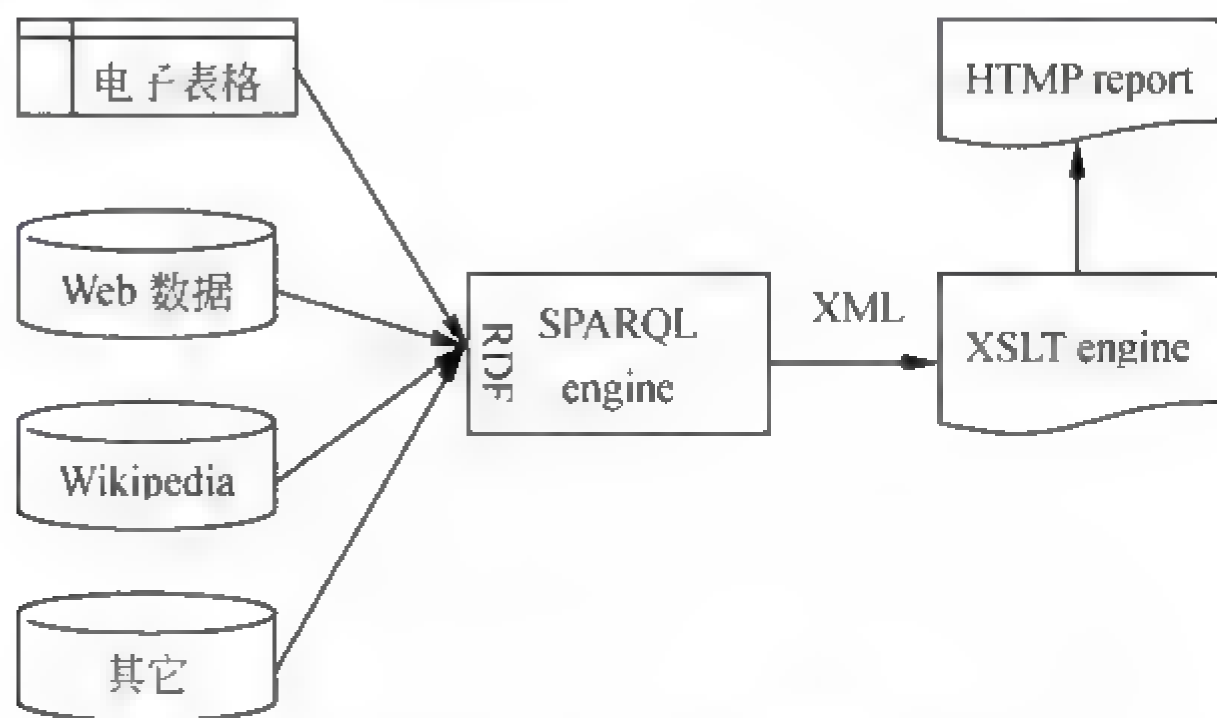

图 6-41　SPARQL 应用结构

SPARQL 建立在多项关键技术的基础之上，就像 HTTP 和 HTML 依赖于 TCP/IP 这样的更深和更低层系统一样。RDF 可以描述任何事物，包括它自身，因此可以从很小的一层开始逐渐丰富。这种薄层方法用于建立词汇栈。目前，RDF 之上的层包括 RDFS 和 OWL。RDFS 即 RDF Schema 语言，它为 RDF 添加类和属性。OWL（Web 本体语言）扩展 RDFS，提供了一种更丰富的语言来定义类之间的关系。更丰富的语言允许使用自动化的推理引擎创建更智能的系统。并且 RDF 特意设计来支持用更抽象的词汇表进行分层扩展。扩展的第一种方式是使用 RDF Schema 或 RDFS。RDFS 为 RDF 增加了类、属性和继承的特性，几乎是面向对象设计者完备的工具箱。OWL 在 RDFS 的基础上提供了及其丰富的工具箱来描述类的属性和关系，OWL 提供了大量的属性来准确描述两个类之间关系的特点，OWL 允许告诉推理引擎在本体中定义的各种属性之间隐含的等价意义。OWL 的主要动机是为本体的语义打下坚实的基础，使推理引擎能够对数据进行自由演绎。

SPARQL 查询返回的结果采用 XML 格式，封装了和查询匹配的变量。很容易提取 XML 信息并显示在 Web 上。同时，SPARQL 允许从 RDF 数据库（或者三元组库）中查询三元组。表面上看和从关系数据库中提取数据的结构化查询语言（SQL）非常类似。但三元组库和关系数据库是完全不同的东西。关系数据库以表为基础，数据都存储在固定的表中，通过外键定义表行之间的关系。三元组库只存储三元组，描述事物的时候可以使用任意数量的三元组。由于三元组库是一个庞大无序的三元组集合，SPARQL 查询通过定义匹配三元组的模板（称为 Graph Pattern）来完成。同时 RDF 三元组库中的三元组构成了描述一组资源的图（如下程序片段就是图的定义方法）。使用 SPARQL 提取三元组库数据需要定义和图中语句相匹配的模式。而且 RDF 没有外键和主键，它使用的是 URI，万维网的标准引用格式。通过 URI，一个三元组库可以直接链接到任何三元组库的其他任何数据。

```
:JournalGraph rdf:type ja:MemoryModel ;
rdfs:label "GraphName" ;
ja:content [ja:externalContent <file:路径>] ;
ja:content [ja:externalContent <file:路径>];
```

不过，所有的 RDF 定义的都是图，三元组库的配置就是发现数据和将其放入图中，或者指定公开这些图的方式。在上面程序片段中，ja:content URI 就应该指向哪里，并将定义一个内存模型对象 GraphName，链接到两个外部磁盘文件。

SPARQL 提供了 SELECT、ASK、DESCRIBE 和 CONSTRUCT 四种不同形式的查询。SELECT 查询形式用于标准查询，并以标准 SPARQL XML 结果格式返回查询结果；ASK 的结果是 yes/no，没有具体内容；DESCRIBE 用于提取本体和实例数据的一部分；CONSTRUCT 根据查询图的结果生成 RDF。如下程序片段是 SPARQL 查询语法：

```
PREFIX : <URI>
SELECT ?notes
WHERE
 {
 ?e a :JournalEntry.
 ?e :notes ?notes.
 ?e :date ?date.
 }
ORDER BY ?date
```

每个 SPARQL SELECT 查询都包括一组按顺序排列的部分。第一部分是序言，包括可选的 BASE 定义和一些前缀定义。其后的 SELECT 部分以 SELECT 开始，描述搜索哪个图的可选的数据集部分，后面用 WHERE 子句表达描述目标结果的图模式。WHERE 子句之后是一些结果修饰符：Order 子句、Limit 子句或者 Offset 子句。

使用图模式获取结果的过程非常简单。多数三元组库都在内存或者数据库中保存三元组，可按照主语、谓语和宾语查询。SPARQL 在查询图模式中用一组三元组表示。首先假设图模式中只有一个三元组，查询可能提供具体的主语 URI。这样的话，三元组库可以忽略没有该主语的所有三元组。然后再筛选掉与图模式提供的谓语不匹配的所有三元组。最后，如果 SPARQL 提供了具体的宾语，还可进一步排除不匹配的三元组。如果查询的主语、谓语或宾语中使用了变量，则不排除那些不匹配的三元组，三元组库将其全部保留，因为可能和变量匹配。如上例中第一个三元组是?e a :JournalEntry.。a 是 rdfs:type 的简写，因此三元组库可以忽略所有谓语不是 rdfs:type 的三元组，然后再筛选掉宾语不是:JournalEntry 的三元组，余下的就是可以作为?e 结果的三元组。

上述例子查询的图模式包含多个三元组，因此三元组库在完成之前还需要对其他三元组做同样的处理。如果一个变量出现在多个位置，则可以使用所有那些值相同的三元组的交集。上例中所有匹配的三元组必须有一个匹配的主语。如果不符合，则丢弃，结果中只保留剩下的三元组。变量匹配可能有多种方式，所以会有多个结果。三元组库的最后一步是根据结果集需要的变量创建结果集。在搜索的最后，三元组库将得到包含?e、?notes、?date 的结果集，这些都是查询中定义的变量。如果 SELECT 查询的形式为“SELECT ?date ?notes”，则不返

回?e。

支持在 Jena 中使用 SPARQL 可以通过称为 ARQ 的模块得以实现。除了实现 SPARQL 之外，ARQ 的查询引擎还可以解析使用 RDQL 或者它自己内部的查询语言表示的查询。ARQ 的开发很活跃，但它还不是标准 Jena 发行版本中的一部分。但是，可以从 Jena 的 CVS 仓库或者自包含的下载文件中获得它。

命令行 sparql 工具对于运行独立查询有用，同时 Java 应用程序也可以直接调用 Jena 的 SPARQL 功能。通过 com.hp.hpl.jena.query 包中的类，使用 Jena 来创建和执行 SPARQL 查询。使用 QueryFactory 是最简单的方法，QueryFactory 有各种 create()方法，用来从文件或者 String 读取文本查询。这些 create()方法返回 Query 对象，这个对象封装了解析后的查询，下一步就是创建 QueryExecution 的实例，这个类表示查询的一个执行。要获得 QueryExecution，需要调用 QueryExecutionFactory.create(query, model)，并传入要执行的 Query，以及查询要处理的 Model。因为查询的数据是编程方式提供的，所以查询不需要 FROM 子句。QueryExecution 上有几种不同的执行方法，每个方法执行一种不同类型的查询。对于简单的 SELECT 查询，可以调用 execSelect()，该方法将返回 ResultSet。ResultSet 支持在查询返回的每个 QuerySolution 上进行迭代，这提供了对每个绑定变量值的访问。另外，还可以使用 ResultSetFormatter，以不同的格式输出查询结果。如下程序实现查询的示例语法片段：

```
InputStream in = new FileInputStream(new File("查询文件名.rdf"));
Model model = ModelFactory.createMemModelMaker().createModel();
model.read(in,null);
in.close();
String queryString =
    "PREFIX foaf: <URI> " +
    "SELECT ?url " +
    "WHERE {" +
    "?contributor foaf:name \"名称 \" . " +
    "?contributor foaf:weblog ?url . " +
    " }";
Query query = QueryFactory.create(queryString);
QueryExecution qe = QueryExecutionFactory.create(query, model);
ResultSet results = qe.execSelect();
ResultSetFormatter.out(System.out, results, query);
qe.close();
```

在 SPARQL 的术语中，这些查询针对在使用 Jena 的 API 时，通过某个查询的 FROM 子句、sparql 命令的，以及 data 开关或者通过向 QueryExecutionFactory.create()传递一个模型来指定的图。除此之外，SPARQL 还能查询任意数量的图，它是根据它们的 URI 来识别的，而这些 URI 在一个查询内是互不相同的。但在研究使用命名图的方法之前，需要解释一下如何向查询提供这些 URI。它可以在查询内部用 FROM NAMED <URI>指定，在这里，是通过 URI 来指定图。另外，也可以用-named URL 把命名图提供给 sparql 命令，URL 用于指定图的位置。最后，可以用 Jena 的 DataSetFactory 类指定要用编程方式查询的命名图。在 SPARQL 查询内部用 GRAPH 关键字调用命名图，后面是图的 URI 或变量名，这个关键字后面是要与

图匹配的图形模式。当 GRAPH 关键字和图的 URI（或者已经绑定到图的 URI 的变量）一起使用时，图形模式就被应用到这个 URI 标识的任何图。如果在指定的图中发现匹配物，那么该匹配物就成为查询结果的一部分，程序片段如下：

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?name
FROM NAMED <jon-foaf.rdf>
FROM NAMED <liz-foaf.rdf>
WHERE {
  GRAPH <jon-foaf.rdf> {
    ?x rdf:type foaf:Person.
    ?x foaf:name ?name.
  }.
  GRAPH <liz-foaf.rdf> {
    ?y rdf:type foaf:Person.
    ?y foaf:name ?name.
  }.
}
```

在程序中，有两个 FOAF 图被传递给查询。查询结果是在两个图中都可以发现的那些人的名字。表示每个 FOAF 图的那些人的结点是空结点，它们的作用范围有限，只在包含它们的图中有效。这意味着表示同一个人的结点不能在查询的两个命名图中同时存在，所以必须用不同的变量（x 和 y）来表示它们。

Jena 的设计目标是可以良好地处理 RDF 数据模型，正如 JDBC 适合处理关系模型一样。数据库应用程序中编写的大量代码都用来保存 Java 对象，还有一些代码用来从数据库中聚集对象。即必须实现 Java 对象和 RDF 之间的相互转换。这时，Google 提供了一种 Java 与 RDF 转换的 Java-to-RDF 工具[①]。它简化并减少所需的代码量，并实现了将一个 bean 保存为 RDF、将其属性与特定的 RDF 属性绑定、将其与其他对象关联和再次回读 bean。

Jenabean 提供了许多功能来定制 bean 如何序列化为 RDF。如果默认设置符合要求，那么就可以开始快速编写和读取 bean。下面创建一个简单的 JavaBean 示例，使它满足所有必需的要求。正如使用 Java Persistence API (JPA)或 Hibernate 一样，需要保证对象有唯一的 ID。Jenabean 需要将一个单独的注释— @Id —添加到至少一个 bean 字段，使用它充当唯一标识符。即@Id 帮助 Jenabean 为每个 JavaBean 实例创建一个唯一的 URI。应该将它放到返回一个唯一的 int 或 String 的 getter 方法中。@Namespace 被应用到类级别中。它覆盖默认的行为，允许定义希望使用的命名空间。其值必须是一个有效的 URI，以/或#结束。@RdfProperty 将 JavaBean 属性与任意的 RDF 属性相绑定。只需要注释 getter 方法并将 RDF 属性 URI 作为参数提供给注释。

Bean2RDF 是一个将对象作为 RDF 编写的 Jenabean 类。它默认情况下是 shallow 模式，这意味着它将保存实例和其单一属性，程序片段分别如下：

---

① http://code.google.com/p/jenabean

一个简单的 bean：

```
package example;
import thewebsemantic.Id;
public class Person {
   private String email;
      @Id
      public String getEmail() { return email;}
      public void setEmail(String email) { this.email = email;}
}
```

在上面程序中的 Jenabean 提供足够的信息来可靠地保存和装载 Person 类实例。没有必要扩展任何内容或编写 XML 描述符文件。由于电子邮件地址是唯一的，它是有效的 ID。

使用生成的 RDF 保存 Person 类的实例：

```
Model m = ModelFactory.createOntologyModel();
Bean2RDF writer = new Bean2RDF(m);
Person p = new Person();
p.setEmail("person@example.com");
writer.save(p);
m.write(System.out, "N3");
...
<http://example/Person>
     a owl:Class ;
<http://thewebsemantic.com/javaclass>
     "example.Person".
<http://example/Person/taylor cowan@yahoo.com>
     a  <http://example/Person> ;
<http://example/email>
     "taylor_cowan@yahoo.com"^^xsd:string
```

在程序中，如果 Person 类还没有添加到模型中，它将断言一个新类作为 owl:Class 的实例。在上面程序代码中 Jenabean 使用 example 包作为一个新的本体类的命名空间。第二个断言是一个注释，指明用于创建个体的 Java 类。Person 实例及电子邮件地址都进行了断言。Jenabean 首先为已保存的实例创建 URI。它还处理电子邮件属性并将其断言为一个 string 字母值。

使用 RDF 表示的个体需要一个 URI，然而 Java 开发人员倾向于使用唯一的 ID。Jenabean 通过将声明的 ID 字段附加到命名空间（这种情况下默认来自包和类名）来提供帮助。因此，Jenabean 针对所有原语类型及其包装程序继承了 Jena 在 Java 和 RDF 之间的类型映射默认值。Jenabean 具有对 java.util.Date 的额外支持，它映射到 xsd:dateTime。数组属性被映射到 rdf:Seq，但表示多重性的最自然的方式是使用 java.util.Collection<?>接口。例如，如果 Person 类有许多 Appointment，将赋给它一个 java.util.Collection<Appointment>类型的属性。Jenabean 不支持具体的 java.util.Collection 实现，比如 List 或 ArrayList，因为 RDF 不会保证顺序。当然也

可以具有其他 bean 的属性：可以是一个单一的实例、一个数组或一个集合，但是也可以使用 RDF2Bean 从模型中装载信息：

```
RDF2Bean reader = new RDF2Bean(m);
Person p2 = reader.load(
   Person.class,"person@example.com");
...
Collection<Person> people = reader.load(Person.class);
//Jenabean 也可以装载所有的 Person 实例
```

这些是在模型中访问 bean 的最简单方法。Jenabean 还支持到 SPARQL（RDF 的 SPARQL 查询语言）结果的绑定。简言之，Jenabean 至少要求 bean 作者指明哪个字段保存的值对于该类型的所有实例是唯一的。保存了 bean 后，将根据类的包和名称为 bean 的类和属性提供默认的 URI。这允许开始从 Java 层轻松地快速创建 RDF。

Jenabean 还支持声明使用的命名空间，即可以使用@Namespace 注释将 bean 映射到特定的命名空间，程序示例代码如下：

```
@Namespace("http://jenabean.googlecode.com/")
public class Person {
<http://jenabean.googlecode.com/Person/person@example.com>
  a    <http://jenabean.googlecode.com/Person> ;
```

这里为 Person 类及其属性（而不是默认包）提供了新的命名空间，该命名空间与作为@Namespace 注释的参数提供的命名空间匹配。默认情况下，这个命名空间将会用于类及其属性。

在 OWL 和 RDF 世界中，通过对同一属性的多个断言来表达各种基数的关系。Jenabean 通过使用 java.util.Collection 接口极大简化了这一过程。同时，通过 Bean2RDF 来实现，程序片段如下：

扩展 Person 以支持朋友关系：

```
public Collection<Person> friends = new
  LinkedList<Person>();
@RdfProperty("http://xmlns.com/foaf/0.1/knows")
public Collection<Person> getFriends() { return friends;}
public void setFriends(Collection<Person> friends) { this.friends = friends;}
```

使用生成的 RDF 表示朋友关系：

```
Model m = ModelFactory.createOntologyModel();
Bean2RDF writer = new Bean2RDF(m);
Person p = new Person();
p.setEmail("person@example.com");
Person f1 = new Person();
f1.setEmail("friend1@example.com");
```

```
Person f2 · new Person();
f2.setEmail("friend2@example.com");
p.getFriends().add(f1);
p.getFriends().add(f2);
writer.save(p); //modifies the Jena model
m.write(System.out, "N3");
...
foaf:knows
   jb:friend2@example.com, jb@friend1@example.com
```

6.8.5　本体编辑工具：Protégé

Protégé[①]是一个史丹佛大学开发的本体编辑和知识获取软件，开发语言采用 Java，属于开放源码软件。由于其优秀的设计和众多的插件，使其已成为目前使用最广泛的本体论编辑器之一。而第一个 Protégé 系统是 1987 年开发。它采用的推理机是 pellet[②]，也是基于 Java 的开放源码系统。目前，Pellet 支持 OWL 1.1。是由一种基于 Tableau 算法的描述逻辑推理机，由美国马里兰大学（College Park 分校）的 MindSwap 实验室开发，推理结果如图 6-42 所示。

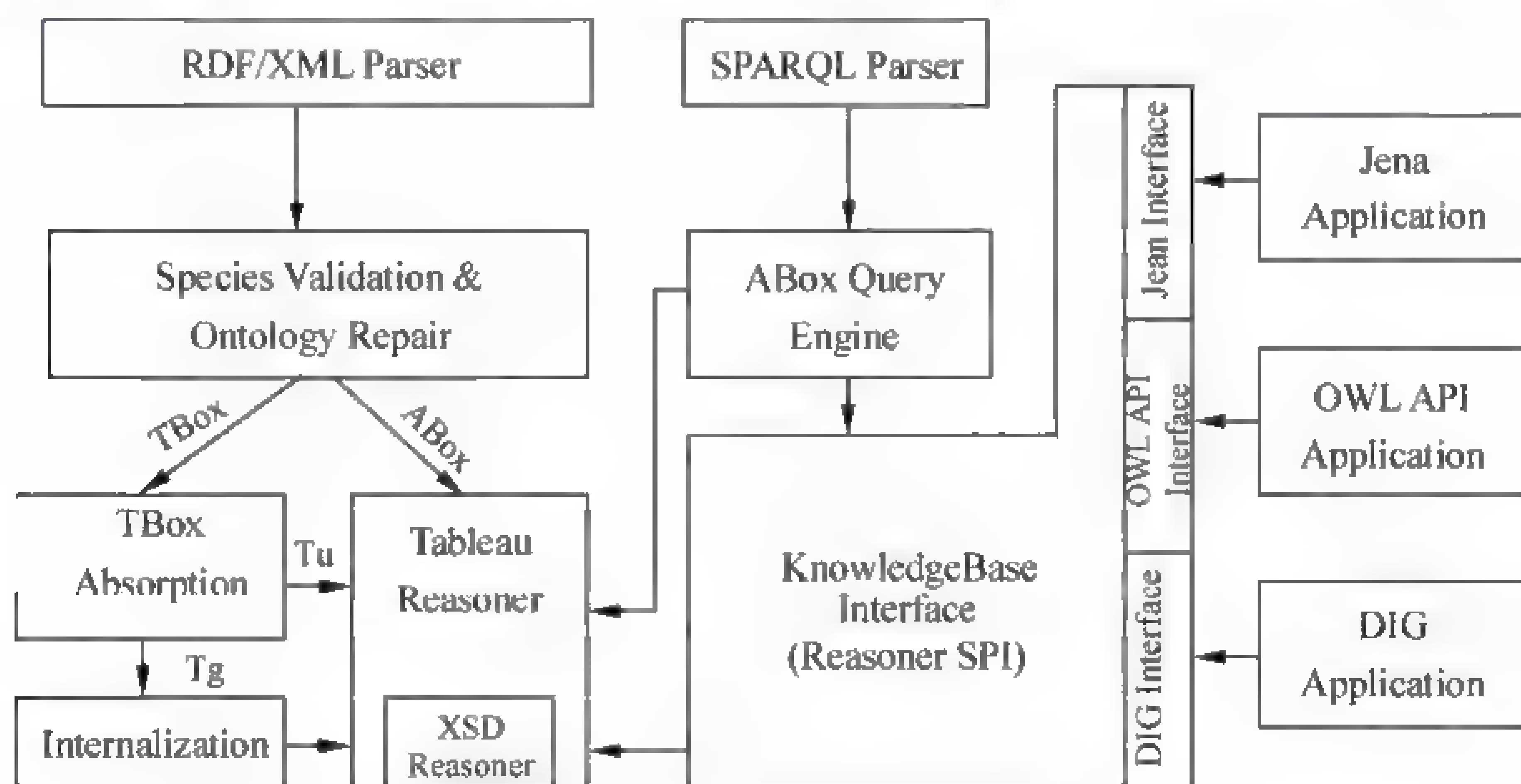

图 6-42　Pellet 推理机推理结构

（来源：http://www.mindswap.org/2003/pellet）

Protégé 本体编辑工具运行如图 6-43 所示，在图中，共有 OWL Classes（OWL 类）、Properties（属性）、Forms（表单）、Individuals（个体）、Metedata（元类）等标签，也可以在“Project”中选择“Configure”项选择其他需要应用的标签，如图 6-44 所示。

通常首先选择 OWL Classes 来编辑，然后建立属性等操作；即可以根据本体的定义进行操作，并根据自己定义的本体进行编辑，以具体的需求实现推理，如图 6-45 所示是 Protégé

① http://protege.stanford.edu

② http://www.mindswap.org/2003/pellet

建立关系的对话框。

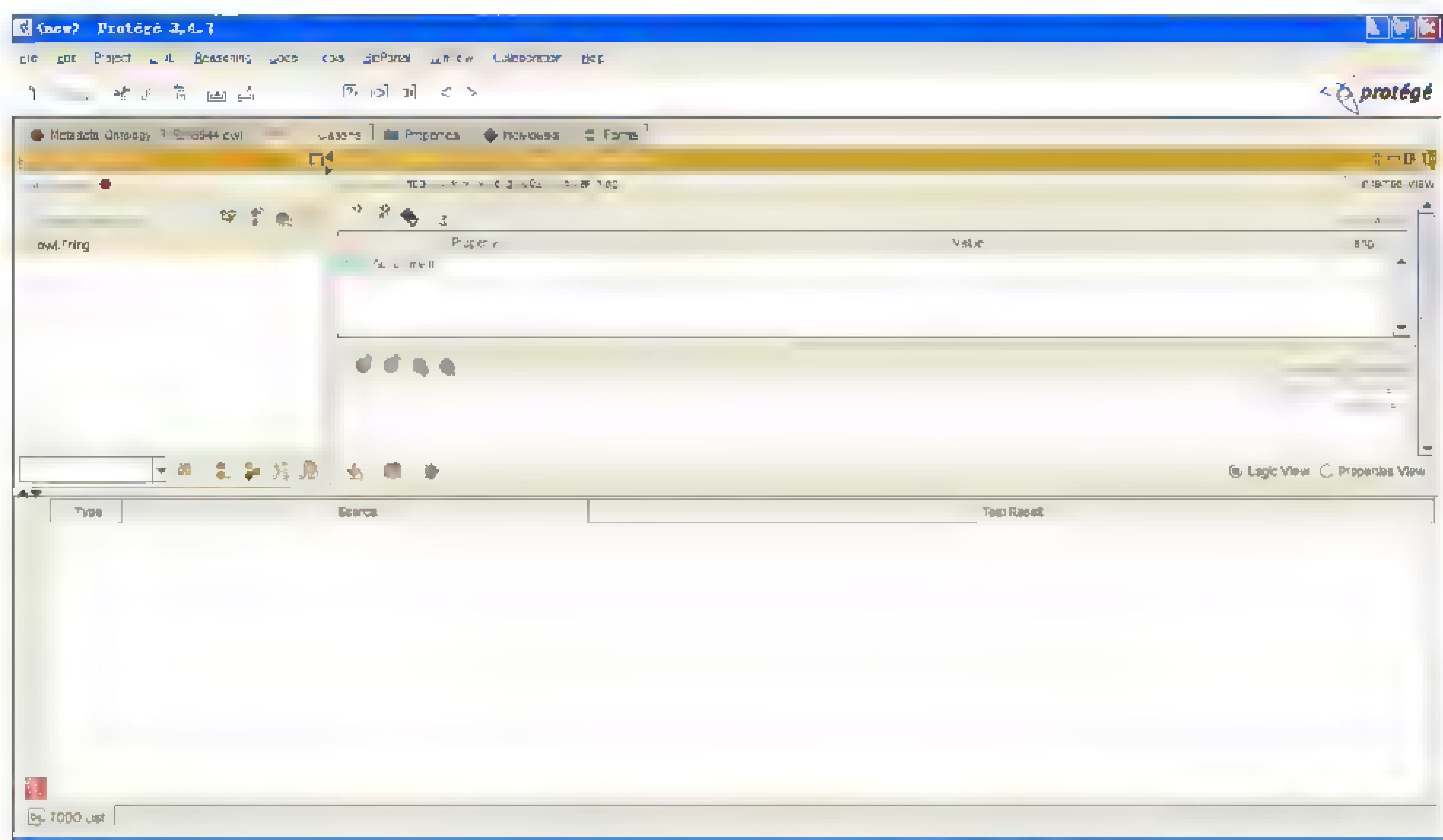

图 6-43　Protégé 本体编辑工具

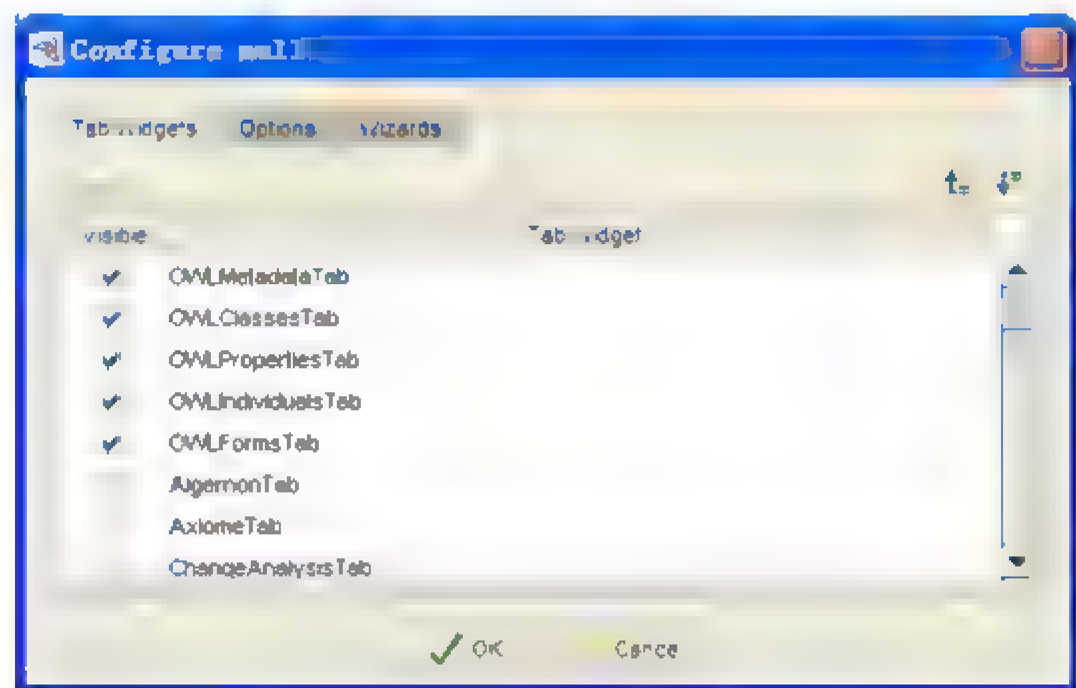

图 6-44　Protégé 标签配置对话框

图 6-45　在 Protégé 在建立关系的对话框

同时，Protégé 也使用 DL Implementation Group（DIG）[①]实现推理，它是一个标准的 XML 描述接口，提供了 XML Schemas、Java XMLBeans 的 Schemas 解析、创建和操纵实例，以及用于诸如 FaCT++ 和 Racer 的 Java Reasoners API 推理。如下代码是 DIG1.1 的一个例子[②]：

```
<?xml version="1.0" encoding="UTF-8"?>
<tells uri="" xmlns="http://dl.kr.org/dig/2003/02/lang"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://potato.cs.man.ac.uk/dig/level0/dig.xsd">
   <equalc>
      <catom name="owl:Thing"/>
      <top/>
   </equalc>
   <defconcept name="ParmezanTopping"/>
   <equalc>
      <catom name="ParmezanTopping"/>
      <and>
         <catom name="CheeseTopping"/>
         <some>
            <ratom name="hasSpicyness"/>
            <catom name="MildSpiciness"/>
         </some>
      </and>
   </equalc>
   <disjoint>
      <catom name="ParmezanTopping"/>
      <catom name="MozzarellaTopping"/>
   </disjoint>
   <disjoint>
      <catom name="ParmezanTopping"/>
...
```

6.8.6 WSMO 编辑工具：WSMO Studio

WSMO Studio 是一个语义 Web 服务和语义业务流程建模环境的 Web 服务建模本体，它提供了一套 Eclipse 插件，图 6-46 是 Eclipse 外挂结构，它也是 GNU Lesser General Public Licence（LGPL）协议下的开源软件。图 6-47 所示是 WSMO Studio 结构。

WSMO 的研究可谓自成体系，ESSI（Enterprise System Solutions, Inc）组织完全抛弃了 W3C 推荐的 OWL（Web 本体语言标准），新定义了 WSML 语言（Web 服务模型语言）和 WSMX 体系结构（Web 服务执行环境）作为对 WSMO 的支撑。目前 WSMO 工具集的研究，主要包括 2005 年 6 月推出的基于 WSMO 的语义 Web 服务编辑器 WSMO Studio；2005 年 6

① http://dig.sourceforge.net
② http://protege.stanford.edu/plugins/owl/api/ReasonerAPIExamples.html

月推出的根据 WSMO 建立语义 Web 服务应用 API 和实现参考的 WSMO4J；2005 年 11 月推出的关于 WSML 语言的一系列工具，如 WSML 推理工具 WSMLDL Reasoner 和 WSML 语言校验工具 WSML Validator。

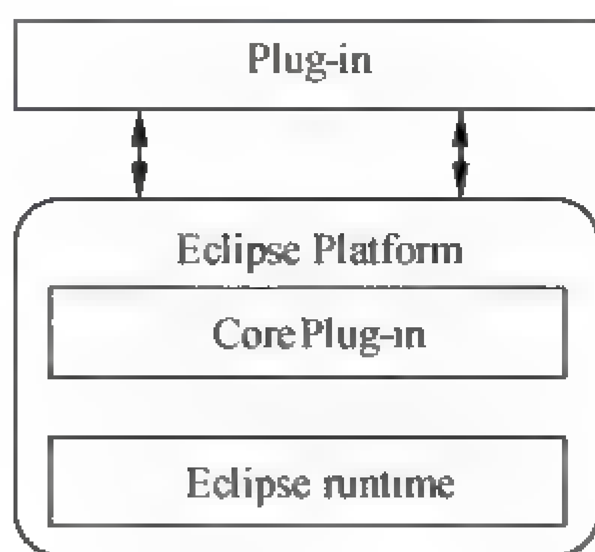

图 6-46 Eclipse 外挂结构

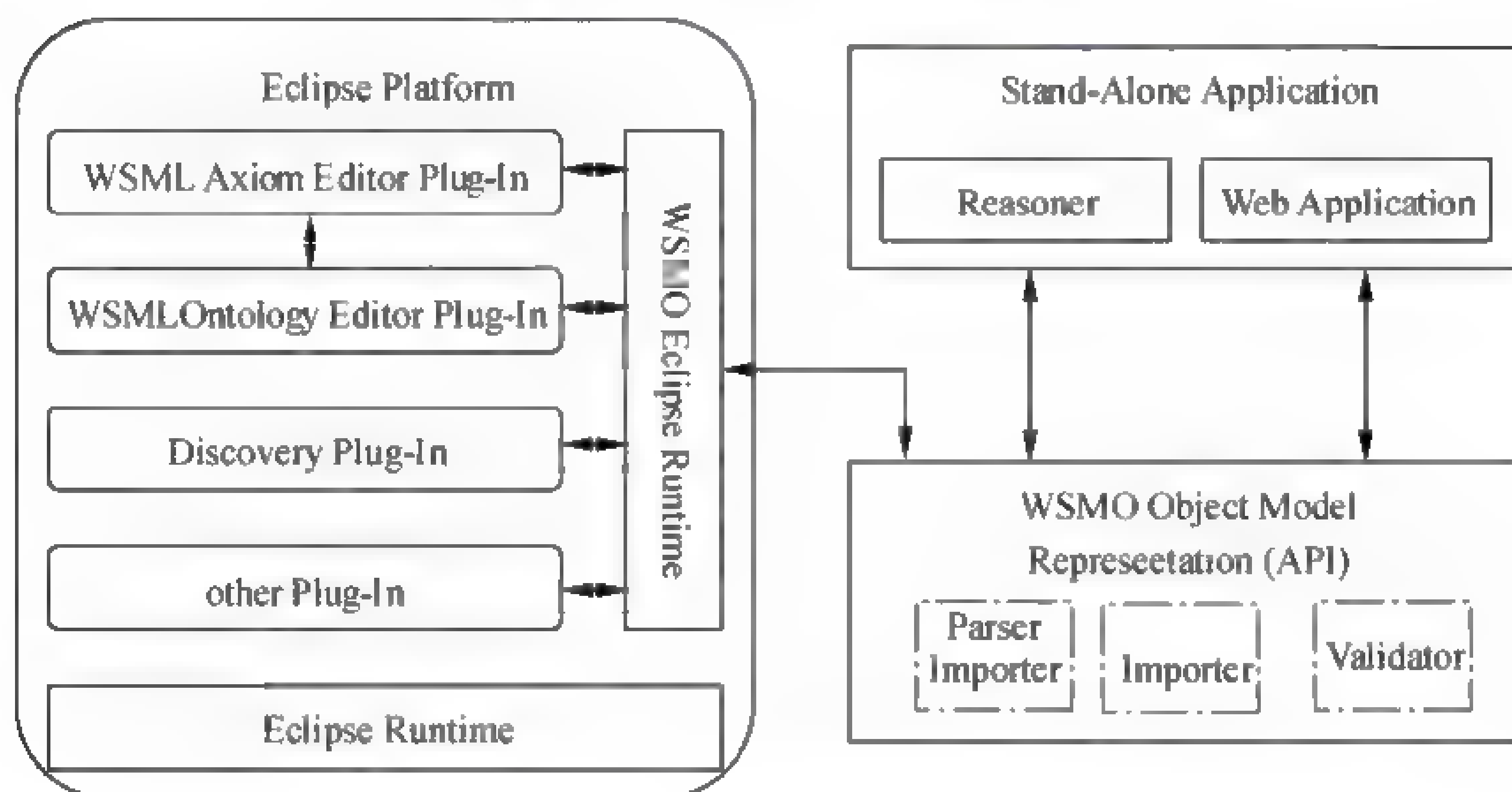

图 6-47 基于 Eclipse 的 WSMO Studio 结构

WSMO Studio 特性主要包括：

（1）本体编辑器集成了 WSML Reasoner（MINS、KAON2、Pellet、IRIS）为本体的一致性检测和查询，编辑器也编辑 WSMO 元素（web services, goals, mediators）。

（2）SAWSDL 编辑器为 WSDL 增加语义注释，而语义商业流程模型根据商业流程模型本体而完成，而且编排设计器为 WSMO 中心编排，从而导入输入 WSML、OWL－DL 子集、RDF、WSML 的 XML 表达，基于 QoS 的服务发现组件，集成 WSML 验证器，语法做色的 WSML 文体编辑，以及基于公理编辑器的 Eclipse GEF，ontology / service / goal 前端可以集成 ORDI 知识库、IRS-III 适配器和 WSMX 适配器。图 6-48 所示是 WSMO Studio 运行结果。具体操作向导可参与 http://www.wsmostudio.org/doc/wsmo-studio-ug.pdf。

6.8.7 SOA 框架：Tuscany

Tuscany 是平台无关的可嵌入框架，可以在各种 Hosting Platform 上运行，如 Tomcat，

JBoss，WAS 等 Web 容器上运行，也可以在 J2SE 环境下运行。Tuscany 的核心模块提供了 SCA 规范的 API 实现。Tuscany 系统提供了的 SPI（Single Program Initiation）、一些系统基本实现（如事件，工厂类，存储等），以及一整套扩展机制，并且这些扩展机制为 Tuscany 整合各个平台的服务提供了基础。同时，Tuscany 的扩展是完全松散耦合的，框架本身提供了大量的扩展实现，用户也可以在自己的系统中扩展 Tuscany 的实现，只需要遵循 Tuscany 的扩展规范以及 API 接口，如图 6-49 所示。

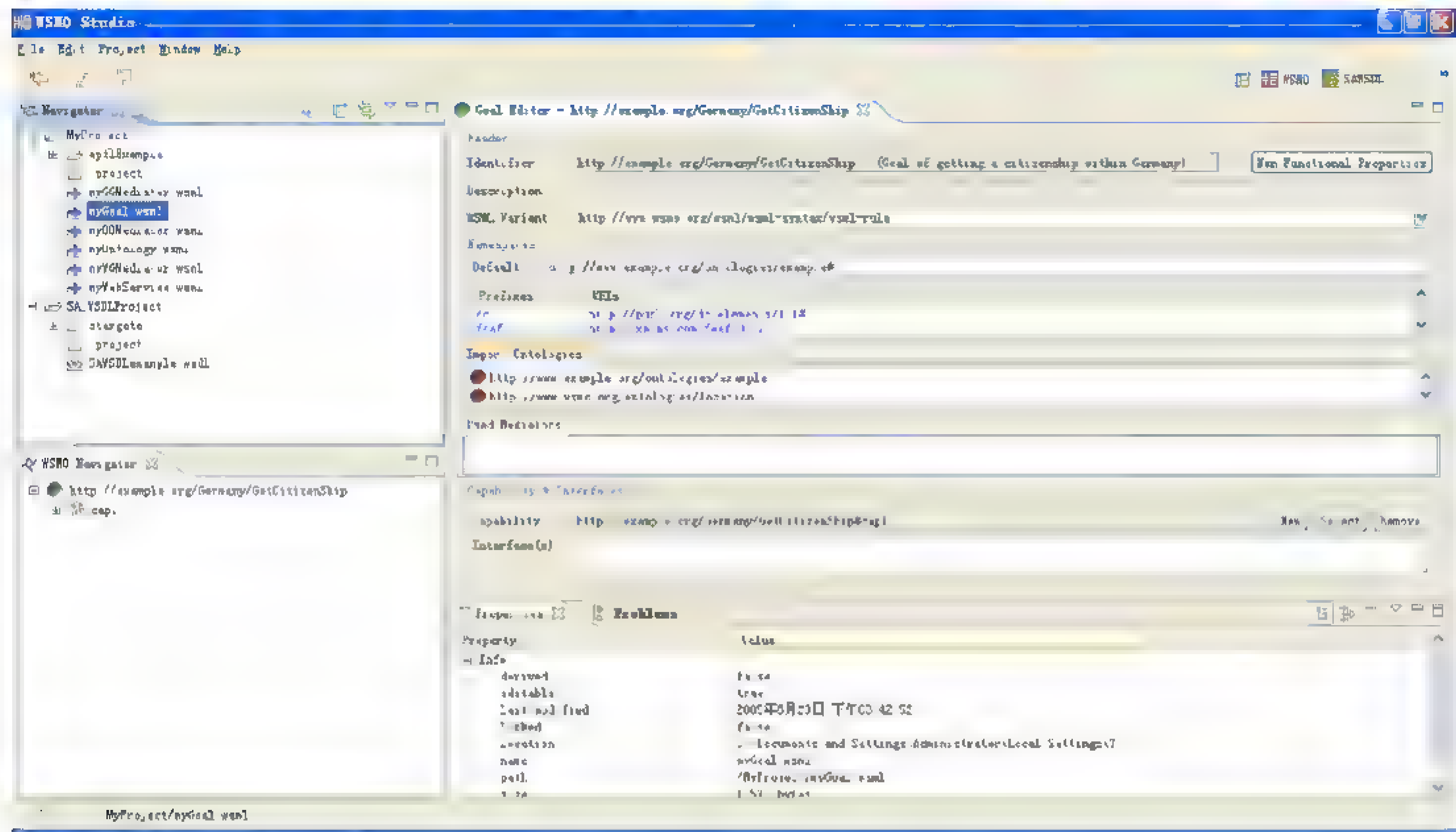

图 6-48　WSMO Studio 运行情况

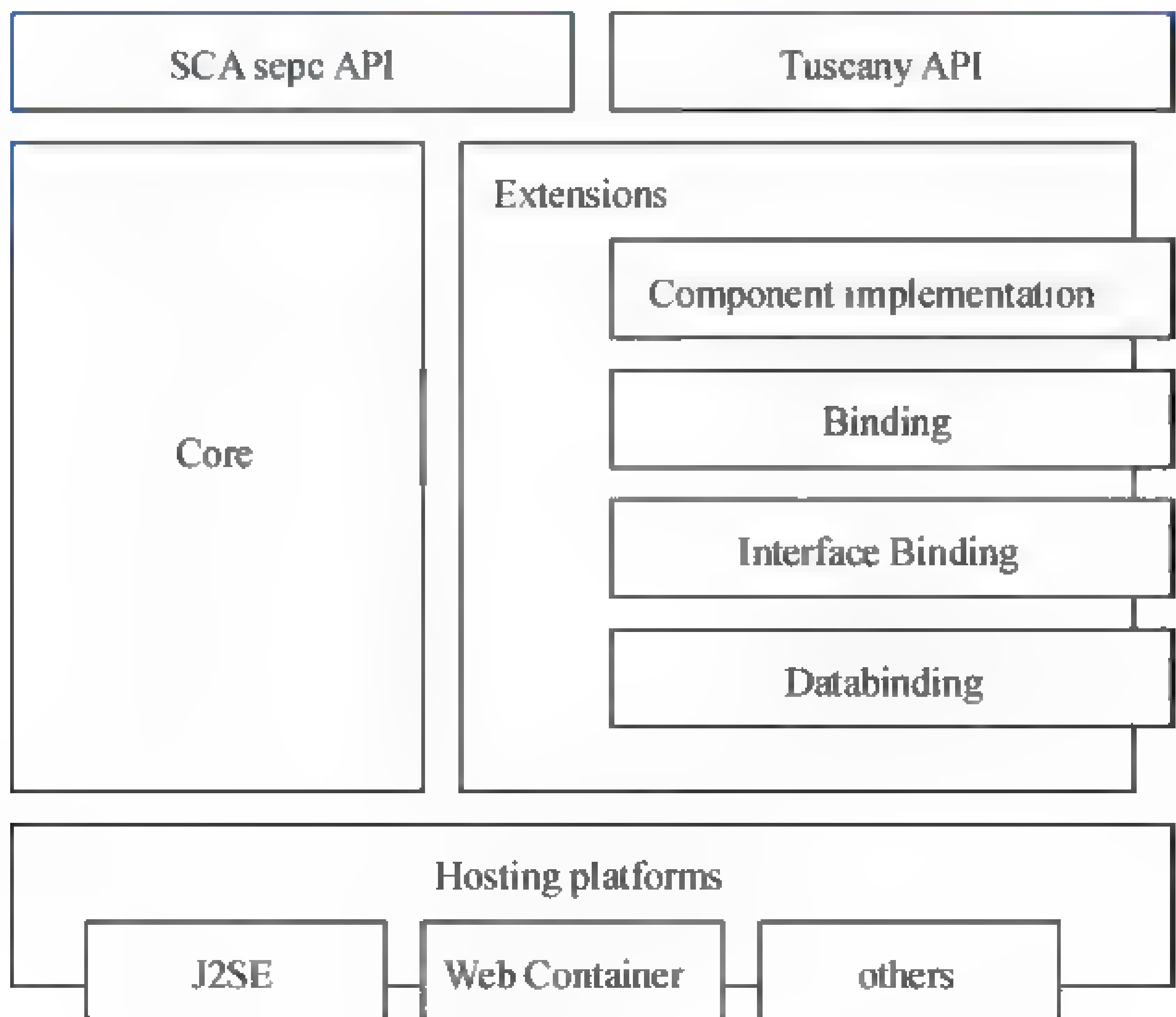

图 6-49　Tuscany 结构

6.8.7.1 Apache Tuscany 应用基础

Apache Tuscany [①]提供全方位的开源 SOA 基础架构以利于开发、组装、发布、管理构件式应用服务（Composite Applications）及数据处理。该项目实现了 SCA 和 SDO 等 OASIS OpenCSA（http://www.oasis-opencsa.org/）标准。Apache Tuscany 提供开放式可扩展的运行环境以支持现在和将来的各种技术，这将解除应用程序对底层技术的依赖和耦合，使得跨技术网络平台的组装成为可能并大大简化：

（1）多种构件实现，包括 Java、BPEL、Xquery、JavaScript；

（2）多种通信协议，包括 RMI、Web Services、JSONRPC、Feed、EJB、CORBA；

（3）多种接口语言，包括 Java、WSDL；

（4）多种数据绑定，包括 XML、JavaBeans、JAXB、SDO、 XMLBeans、JSON、AXIOM。

Apache Tuscany 集成 OSGi、Spring、JEE 和 Web 2.0。该项目提供了从小型到企业级业务的广泛支持。使得解决方案提供商、中间件平台提供商和最终用户和开发人员都可获益。Tuscany 是一轻量级平台，可以独立运行或嵌入在 WebSphere（IBM）、Geronimo、Tomcat 和 Jetty 等应用服务器中。Tuscany 在 Web 2.0 方面主要提供了以下几类扩展：

（1）Implementation 扩展：script implementation 提供了各种脚本语言的实现，如 Javascript、python、ruby 等，widget implementation 提供了将一个 SCA Component 封装成 widget 的能力，resource implementation 提供了一种简单的 http 资源的实现。

（2）Binding 扩展：atom binding 提供了 atom 方式的绑定实现，dwr binding 提供了利用 DWR 框架进行 ajax 调用的能力，http binding 提供了直接进行 http 访问的能力，jsonrpc binding 提供了在 Javascript 中使用 jsonrpc 进行 AJAX 调用的能力。

（3）Databinding 扩展：JSON 格式的 databinding 提供了将 JSON 格式的数据与其他格式（如 xml，Java Bean，SDO 等）之间互相转换的能力。

SCA 组合应用程序中的组件可以在网络中的不同结点上运行，而在 Apache Tuscany 中，可以使用 SCA 域管理一组结点，图 6-50 所示是 Tuscany 一种应用模式。在 SCA 中，组合、组件、其实现和运行它们的结点属于一个所谓的 SCA 域。诸如 Tuscany 等 SCA 实现提供了管理工具，允许系统管理员管理域中的 SCA 构件。使用域可以提供在将结点添加到域时指定结点安装特征（例如主机和端口）的灵活性，而不是在组合文件中指定这些特征。同时，域中的所有 SCA 资源——贡献包、组合和结点全都是可以通过 HTTP 进行访问的 Web 资源。这些资源的集合可通过 Atom 进行访问，并且可以使用 Atompub 进行管理。当包装 Tuscany 运行时，该运行时由 Tuscany 分发包库组成，每个 SCA 可部署组合在一个结点中运行，每个结点使用某个贡献包、某个组合和运行结点的环境的属性（例如主机和端口）进行配置。如下程序片段是添加结点的配置方法：

```
<?xml version='1.0' encoding='UTF-8'?>
<entry xmlns="http://www.w3.org/2005/Atom">
 <id>
```

① http://tuscany.apache.org/chinese-portal.html

```
composite:http://tuscany.apache.org/cloud;http://tuscany.apache.org/cloud;
MyCatalogsNode
 </id>
 <content type="text/xml">
  <composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
   xmlns:t="http://tuscany.apache.org/xmlns/sca/1.0"
   targetNamespace="http://tuscany.apache.org/cloud"
   xmlns:c="http://services"
   name="MyCatalogsNode">

   <component name="MyCatalogsNode"">
    <t:implementation.node uri="http://myws" composite="c:catalogs"/>
    <service name="Node">
     <binding.ws uri="http://localhost:8081">
     <t:binding.http uri="http://localhost:8081"/>
     <t:binding.jsonrpc uri="http://localhost:8081"/>
     <t:binding.atom uri="http://localhost:8081"/>
    </service>
   </component>
  </composite>
 </content>
</entry>
```

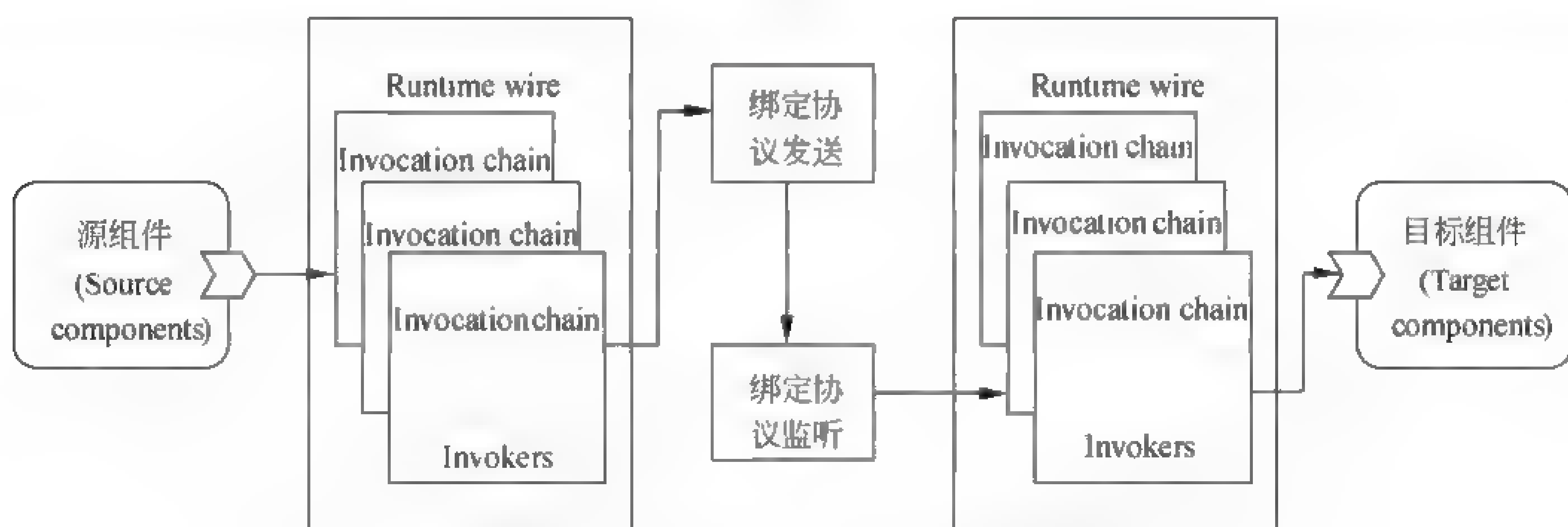

图 6-50　面向 SCA 的 Tuscany 应用模式

而当在 Tuscany 中用 Java 实现 SCA 构件组合时，需要配置具体的 SCA 环境。SCA 的配置主要放在.composite 文件中，它是 SCA 定义的一种 XML 语言服务组件定义语言（Service Component Definition Language，SCDL），一个.composite 文件实质上是一个 SCA Composite 的 XML 配置，这些.composite 文件一般位于 classpath 下的 META-INF/sca-deployables 路径中，Tuscany 的 runtime 在初始化的时候会在 classpath 中搜索所有.composite 文件，并且将它们定义的 SCA Composite 初始化。如下程序代码就是 composite 配置格式，下面是以一个 SCA Composite：TestComposite 实例为例进行说明，并在 src/main/resources/META-INFO/sca-deployables 下建立 XML 文件 test.composite：

```
<?xml version="1.0" encoding="UTF-8"?>
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
    targetNamespace="http://tuscany.demo.kth.org"
    xmlns:tuscany="http://tuscany.apache.org/xmlns/sca/1.0"
    xmlns:dbsdo="http://tuscany.apache.org/xmlns/sca/databinding/sdo/1.0"
    name="TestComposite">
    <component name="testImpComponent">
      <implementation.Java class="org.test.impl.testImp"/>
      <reference name="testOK">
          <interface.wsdl
          interface="http://ws.test.org#wsdl.interface(testOK)"/>
           <binding.ws
             uri="http://localhost:8085/services/ testOKService"/>
        </reference>
        <reference name="testUp">
            <tuscany:binding.rmi host="localhost" port="8099"
                serviceName=" testUpService" />
        </reference>
    </component>
    ...
    <service name="testPE" promote=" testImpComponent / testPE ">
        <interface.Java interface="org.test. testPE "/>
        <tuscany:binding.jsonrpc />
    </service>
</composite>
```

在程序片段中，需要定义的 namespace，解释如下：

（1）tagetNamespace：用于表示此 SCA Composite 的命名空间，在 SCA 部署的时候，contribution 文件中的 composite 部署配置应于这个命名空间一致。

（2）xmlns:tuscany：Tuscany 的特有 Binding 扩展的命名空间，由于 Tuscany 是一个开放式框架，它的 binding 扩展有多种类型，其中有些是 SCA 规范中定义的扩展，这些扩展不需要命名空间，但有些是 Tuscany 自己定义的扩展，这些扩展就需要使用 xmlns:tuscany 定义的命名空间，Web 2.0 扩展大多属于这种类型。用户也可以自己开发扩展，那在使用的时候就需要引入用户自定义的命名空间。

（3）xmlns:dbsdo：Tuscany 的特有 databinding 扩展的命名空间，一般来讲每个 binding 都有自己默认的 databinding，比如 web service 的 binding 就会默认使用 JAXB 的 databinding，大多数情况下不需要特定指出 databinding 的类型，但是不排除某些特殊情况下需要用户特别指出 databinding 的类型，此时就需要加上这个命名空间。

而 SCA Component 的配置，component 元素就是配置一个 SCA Component 的声明，它在同一个 composite 中必须唯一的名称，且每一个 component 都必须有一个 implementation 元素，可以采用 Java 实现 Component，因此需要配置 implementation.Java，这个配置会制定一个 Java 类的全名作为此 Component 的实现。Component 中也可以配置 Service 和 Reference，Service

是定义该 Component 暴露出的一些服务，Reference 是定义该 Component 需要引用的其他外部服务。在 Reference 中，需要指定该引用的接口（interface）和绑定（binding）。

服务 testPE 表示 SCA Component 暴露给 Web 2.0 应用的服务。通过一个外部的 service 定义，该 Component 可以指定一个在 composite 层暴露的服务。此服务的名称必须与用 @Service 指定的接口名一致，在该服务的定义里也必须指定 interface 和 binding，其中 binding 采用 jsonrpc 方式，便于在 web client 端使用 ajax 调用，注意此处必须加上 tuscany 的名称空间。

这样通过这个配置文件将开发的 Java 实现封装成了 SCA Component，并使其可以在 Tuscany 中运行，从而达到开发基于 Tuscany 的应用程序的目的。

当要运行 SCA 业务应用程序时，SCA 运行（比如 Tuscany）首先需要加载并配置 SCA 复合文件（见上述两个文件配置方法）。对每个复合文件工件进行检查，然后使用工厂方法在内存中实例化不同对象。第二步是实例化用来连接组件的运行时连接（runtime wire）。在这个步骤中，将通过复合文件中提到的绑定组件引用和组件服务创建运行时连接。运行时连接是一个调用链集合，并且业务接口中的每个方法都有一个调用链。将创建一个处理程序，它充当一个消息交换台操作程序（switchboard operator），并将每个方法调用传递到相应的调用链。每个调用链由一组调用器（invoker）和拦截器（interceptor）组成。其中调用器为绑定协议和实现技术提供调用逻辑；拦截器是一种特殊的调用器，它提供其他的功能，比如数据转换、安全性和事务控制。对于组件引用，将创建一个运行时连接来通过所选的绑定表示出站（outbound）调用。对于组件服务，将创建一个运行时连接来表示对实现类型的入站（inbound）调用。回调链也被添加到运行时连接中以提供对组件的反向回调。图 6-51 是 SCA 关系图，且是一种基本的应用结构。

在图 6-51 中，当在 Tuscany 运行时中运行，而这个运行时本身又承载在 Apache HTTPD 服务器上。服务器侦听对 SCA REST 服务的 HTTPD 请求，并在接收到此类请求时调用 Tuscany 运行。Tuscany 然后执行相应的组件（由连接决定），并以 HTTP 响应的形式返回结果。这意味着标准 Web 浏览器发出的调用能够调用 SCA 服务，Web 浏览器显示 HTML 页面，其中包含 JavaScript Ajax 代码，Ajax 调用 SCA 组件，并显示所得到的结果数据。同时在图中各组件表示：

（1）RSS/Atom Checker 组件从指定的 RSS 或 Atom Feed 获取最新的文章，并将文章转换为指定的更为简单的 XML 格式。

（2）POP Checker 组件从指定的 POP3 电子邮件账户获取最新的电子邮件，同样也将电子邮件转换为指定的 XML 格式。

（3）NNTP Checker 组件获取 NTTP 服务器上指定新闻组的最新张贴内容。此组件尚未在 Alert Aggregator 示例中实现。

（4）Web Service Checker 组件调用指定的 Web 服务，并将返回的数据转换为简单的 XML 格式。此组件尚未在 Alert Aggregator 示例中实现。

（5）Alert Config 组件管理应用程序的配置，保留关于要检索的 RSS/Atom Feed、要检查的 POP 电子邮件账户等的详细信息。

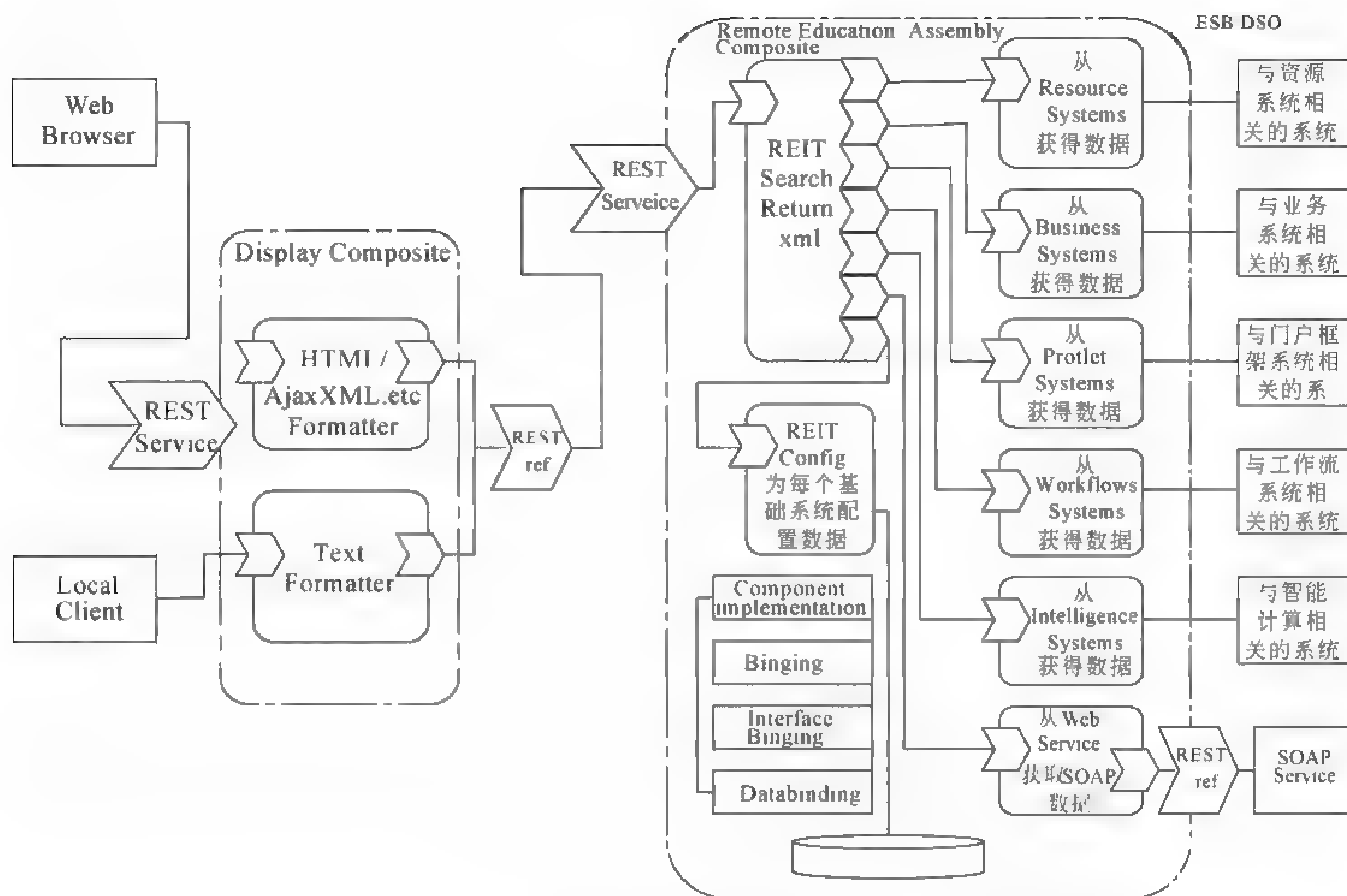

图 6-51 面向 Tuscany 的 SCA 关系

（6）Alert Checker 组件基于 Alert Config 组件提供的配置数据管理对各个 Checker 组件的调用。Checker 组件所返回的 XML 数据聚合为单个 XML 文档，并使用 REST 服务绑定向客户机公开。

而 Display Composite 提供所需的功能将 Alerter Composite 提供的 XML 转换可阅读的格式，如可在 Web 浏览器中显示的 HTML。组合包含以下组件：

（1）HTML Formatter 组件从 Alerter Composite 检索配置 XML 和最新警报 XML，然后基于此数据生成 HTML 表。同时可以通过网页对公开此组件的 REST 服务进行 Ajax 调用来检索此 HTML。

（2）Text Formatter 组件也是从 Alerter Composite 检索 XML 数据，并将其转换为可供人阅读的文本，供本地客户机访问。此组件尚未在 Alert Aggregator 示例中实现。

6.8.7.2 Apache Tuscany 基本应用方法

通过使用常用的 Eclipse 开发环境、Eclipse SOA Tools Platform(STP)插件、Spring 和 Apache Tuscany 可以简化了服务开发。Apache Tuscany 可以与 STP、Spring 集成在一起来为创建的服务提供 SCA 的 Java 程序，使其可以使用 SCA 标准和 Apache Tuscany 注释来注释服务。这里就不详细介绍安装过程了，下面着重叙述 Tuscany、SCA、Spring 整合基本原理和配

置文件的配置方法[①]。

正如前面所述，SCA 提供了一个编程模型，用于创建基于 SOA 的应用程序和解决方案。SCA 所依托的理念是将业务功能作为一系列服务提供，从而创建能满足特定业务需求的解决方案。这些复合集可以包含为已有系统中的应用程序和业务功能创建的新服务，以及作为复合应用的一部分重用的应用程序。

正如前面章节所述 Spring 一个重要的优势在于它的分层架构，它允许选择所使用的组件，同时为 J2EE 应用程序开发提供了一个紧密结合的框架。同时，Spring 为简单的 Java 对象提供了一个框架，从而使它们能够通过包装器类和 XML 配置来使用 J2EE 容器。

开源软件 Apache Tuscany 项目致力于实现 SCA 规范(和一些其他的 SCA 规范，如 Service Data Objects 和 Data Access Service)，以及依照 Open Service-Oriented Architecture（OSOA）和针对全球信息社会（OASIS SCA Java）规范的一些标准，Apache Tuscany 为 SCA 运行时提供了一个全面的基础架构。

另外，OSGi（Open Service Gateway Initiative）有双重含义。一方面它指 OSGi Alliance 组织；另一方面指该组织制定的一个基于 Java 语言的服务（业务）规范——OSGi 服务平台(Service Platform)。OSGi 共由核心规范、标准服务(Standard Services)、框架服务(Framework Services)、系统服务(System Services)、协议服务(Protocol Services)、混合服务(Miscellaneous Services)等几部分共同组成。核心规范是 OSGi 规范中的核心部分，它通过一个分层的框架，实现了 OSGi 最为成功的动态插件机制。同时，OSGi 是服务平台的规范，为 Eclipse 提供了该规范的许多可用实现之一，并用作最新 OSGi R4 规范的参考实现。OSGi 是基于 Java 的框架，旨在用于需要长运行时间、动态更新和对运行环境破坏最小的系统。图 6-52 所示是基于 OSGi 的应用模式。

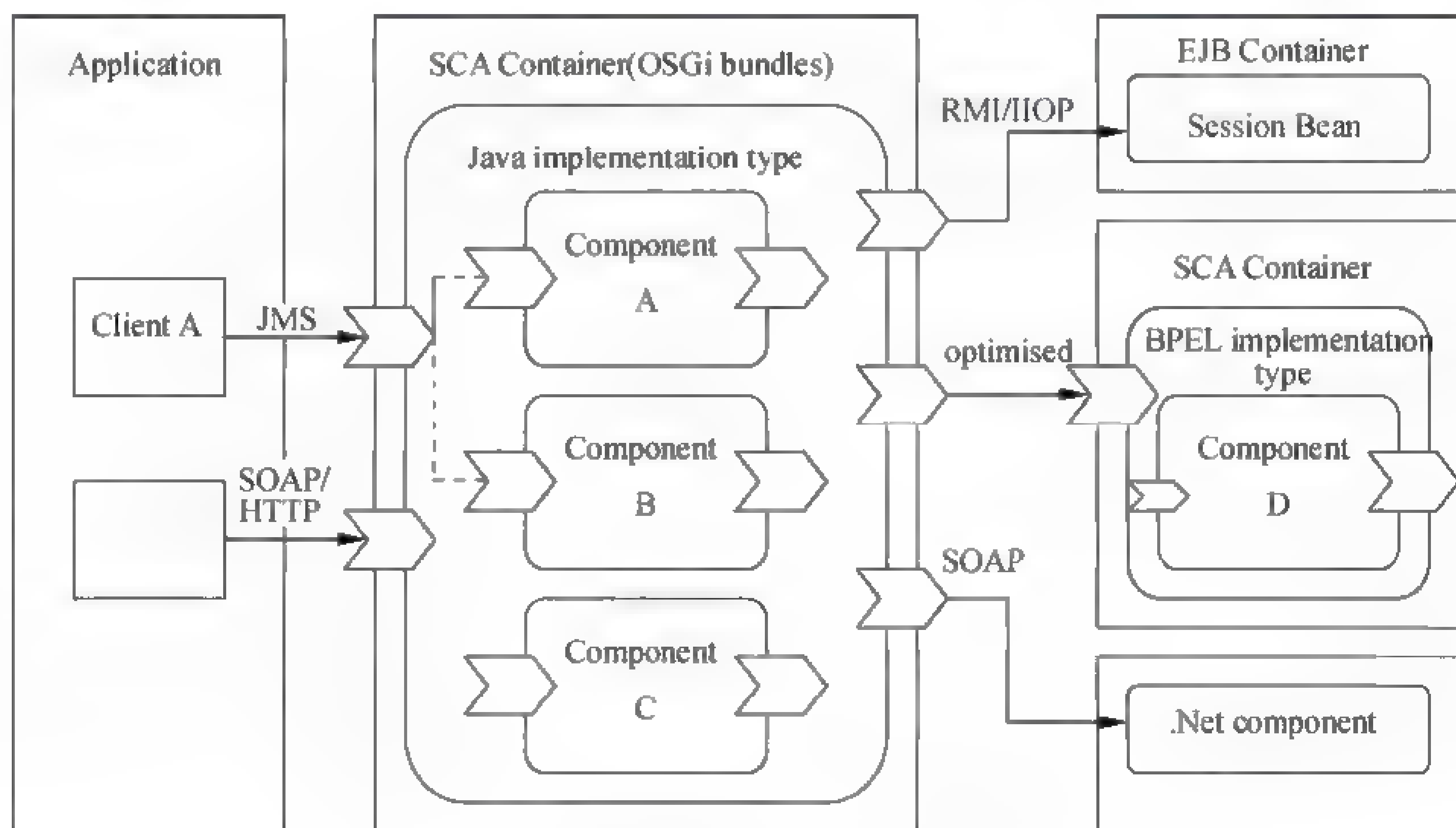

图 6-52　基于 OSGi 的 Tuscany 应用模式

Spring Framework 与 SCA 采用许多相同的设计原则。SCA 将 Spring 视为其组件的一种

① http://www.osoa.org/download/attachments/250/Power_Combination_SCA_Spring_OSGi.pdf?version=3

实现技术。SCA Spring Component Implementation Specification 定义了如何采用这种方式来使用 Spring。与 Spring bean 类似，SCA 组件可以包含到其他组件所提供的服务引用，并且有一些属性可供配置。与 Spring 形成对比的是，SCA 是一种跨语言的分布式组件架构，它支持多种组件通信机制。通过将 Spring beans 发布为可由其他组件访问的服务，并为 Spring beans 提供关联到其他（可能为远程）组件的服务的引用，SCA 可以扩展 Spring 组件的功能。要将 SCA 与 Spring 相结合，一种有效的方法是使用 Spring 来构建粗粒度的服务组件实现，并引入到 SCA 中以便公开服务、关联服务组件，以及处理异构和分布式系统。当在 Apache Tuscany SCA 实现中，SCA 使用 Spring 作为其组件在 SCA 复合集中的实现技术，可以将 Spring 应用程序定义为 SCA 复合集中的 SCA 组件，即 SCDL。

如下程序就是 SCA 与 Spring 整合配置代码：

```
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
     xmlns:tuscany="http://tuscany.apache.org/xmlns/sca/1.0"
     targetNamespace="http://test" xmlns:c="http://test" name="Test">
   <component name="TestServiceComponent">
          <implementation.spring location="targetURI"/>
   </component>
</composite>
```

在程序片段中，<implementation.spring>元素的位置属性可以指定目标 URI 指向某个存档文件（.jar）、目标，或者直接指向 Spring 应用程序上下文，以下描述给出了指定<implementation. spring>位置属性的目标 URI 的可能方法。

（1）指定 Spring 应用程序上下文文件。

```
<implementation.spring location="application- context.xml"/>
```

Apache Tuscany 允许使用多种应用程序上下文来实现 SCA 组件。并且方法是在这个目标应用程序上下文（由此 SCA 复合集文件内的<implementation.spring>元素的 location 属性标识）中定义一个 ClassPathXmlApplicationContext，程序示例代码如下：

```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:sca="http://www.springframework.org/schema/sca"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd
      http://www.springframework.org/schema/sca
      http://www.osoa.org/xmlns/sca/1.0/spring-sca.xsd">
    <bean class="org.springframework.context.support.ClassPathXmlAppl-
    icationContext">
    <constructor-arg>
              <list>
                   <value>context1.xml</value>
                   <value>context2.xml</value>
                   <value>context3.xml</value>
                   ...
```

```
            </list>
        </constructor-arg>
    </bean>
</beans>
```

（2）指定目录（指定存档文件）。目标 URI 将资源指定为名称为 spring 的目录（spring.jar 存档文件），其中包含所有与 Spring 相关的文件。而且 META-INF/MANIFEST.MF 文件必须包含在 Spring 目录中（spring.jar 存档文件中），它使用 Spring-Context :: <path> 格式将 Spring-Context 头部指定到上下文配置的路径。其中，path 相对于 Spring 目录。如果 MANIFEST 文件中没有 MANIFEST.MF 文件或 Spring-Context 头部，则默认行为是使用 Spring 目录下（spring.jar 存档文件中）的 META-INF/spring 目录中的 application-context.xml 文件来建立应用程序上下文。

```
<implementation.spring location="./spring"/>     //指定目录
<implementation.spring location="spring.jar"/>  //指定文件
```

组件实现的业务功能将由其他组件作为服务提供。实现可以依赖其他组件提供的服务，这些依赖关系被称作引用。实现可以有可设置的属性，即影响业务功能运转的数据值。如下程序片段展示了如何将 Spring beans 提供为 SCA 服务，以及如何在 Spring 应用程序上下文中配置 SCA 引用和 SCA 属性：

```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:sca="http://www.springframework.org/schema/sca"
      xsi:schemaLocation="
    http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd
      http://www.springframework.org/schema/sca
      http://www.osoa.org/xmlns/sca/1.0/spring-sca.xsd">
    <bean id="" class=" ">
      <property name=" " ref=" "/>
      ...
    </bean>
</beans>

<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
     xmlns:t="http://tuscany.apache.org/xmlns/sca/1.0"
     targetNamespace="http://test" xmlns:c="http://test" name=" ">
    <component name=" ">
      <implementation.spring location="META-INF/spring/*-context.xml"/>
      <service name=" ">
         <interface.java interface=" "/>
         <t:binding.rmi host="localhost" port="8099" serviceName=" "/>
      </service>
      <reference name " " target=" " />
```

```
        ...
    </component>
    <component name=" ">
        <t:implementation.script script="*.js"/>
    </component>
...
</composite>
```

SCA Spring Component Implementation Specification 和 Apache Tuscany SCA 运行时允许使用 Spring SCA 模式中的自定义 SCA 命名空间元素在 Spring 应用程序上下文文件中声明 SCA 服务、引用和属性。也可以使用自定义 SCA 命名空间元素将 Spring beans 声明为 SCA 服务，并通过 SCA 组件定义指定到所获取的 SCA 服务和属性的引用。使用 Spring 应用程序上下文文件中的 SCA 命名空间元素被称为 SCA 服务、引用和属性的显式声明。

<sca:service>允许将哪些 Spring bean 公开为 SCA 服务，以提供一种方式来控制将哪些 Spring bean 公开为 SCA 服务。SCA 运行时负责创建合适的服务器绑定，根据 SCDL 配置将需要的策略应用到这些服务上。

<sca:reference>提供一种方式来声明 Spring 应用程序上下文对复合集中的其他 SCA 组件所提供的服务的依赖关系。该 SCA 运行时负责创建合适的引用绑定，根据 SCDL 配置将需要的策略应用到这些服务上。

<sca:property>提供一种方式来声明 Spring 应用程序上下文对由 SCA 组件实现提供的可设置属性的依赖关系。<sca:property>元素的 name 属性应该在复合集中拥有一个与所含组件相匹配的 SCA 属性。例如下面的程序示例配置代码：

```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:sca="http://www.springframework.org/schema/sca"
      xsi:schemaLocation="
      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd
      http://www.springframework.org/schema/sca
      http://www.osoa.org/xmlns/sca/1.0/spring-sca.xsd">
<sca:service name=" " type=" " target=" "/>
    <bean id=" " class=" ">
      <property name= " " ref=" "/>
      ...
    </bean>
   <sca:reference name= " " type=" "/>
   ...
</beans>
```

同时，SCA Spring Component Implementation Specification 和 Apache Tuscany SCA 运行时支持直接在 Spring 应用程序上下文文件中声明 SCA 服务、引用和属性，无须使用在 SpringSCA 模式中定义的任何自定义 SCA 命名空间元素。在 Spring 应用程序上下文文件中直

接使用 SCA 引用和属性（无须自定义 SCA 命名空间）称为 SCA 服务、引用和属性的隐式声明。当应用程序上下文中没有显式的<sca:service>元素时，所有顶级的 bean 都将被公开为 SCA 服务，使用 bean 名称作为服务名称。任何内部 bean 或抽象 bean 都不会被用于隐式服务创建。当 Spring bean 实现类实现多个接口时，这些 bean 可以被公开为单个或多个服务。使用显式的<sca:service>元素，其中每个<sca:service>元素引用相同的<bean>元素，但 type 属性仅使用由 bean 提供的接口之一。在隐式创建服务时，bean 被公开为单一服务，方法是将 bean 类本身声明为服务的一个接口。尽管 Apache Tuscany SCA 运行时支持使用隐式 SCA 服务、引用和属性，但也有一些场景不适合使用隐式声明。如下程序片段就是隐式声明为集合示例配置的方法：

```
<bean id=" " class=" ">
  <property name=" ">
    <list>
        <value> </value>
        <ref bean=" " />
    </list>
  </property>
   <property name=" ">
    <map>
        <entry>
            <key>
                <value </value>
            </key>
            <value> </value>
        </entry>
        <entry>
            <key>
                <value>a ref</value>
            </key>
            <ref bean=" " />
        </entry>
    </map>
  </property>
  <property name=" ">
    <set>
        <value> </value>
        <ref bean=" " />
    </set>
  </property>
</bean>
```

同样在隐式声明中，在 Spring 中的构造函数注入允许通过类构造函数注入依赖关系。为了减少潜在的歧义，Spring 建议只要在 bean 实现中定义了多个构造函数，就为<constructor-arg>元素适当使用 index 和 type 属性。Tuscany 还建议为<constructor-arg>元素使

用 index 和 type 属性，为所有构造函数注入场景显式声明相关的 SCA 引用，即使 bean 只有一个构造函数，也可以采用这种引入方法。例如下面的示例程序片段：

```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:sca="http://www.springframework.org/schema/sca"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd
      http://www.springframework.org/schema/sca
      http://www.osoa.org/xmlns/sca/1.0/spring-sca.xsd">
   <bean id=" " class=" ">
    <constructor-arg index="0" type=" " ref=" "/>
     ...
   </bean>
   <sca:reference name=" " type=" "/>
   ...
</beans>
```

最后，在 Spring 中，主要的模块单元是应用程序上下文，它包含一定数量的 bean（由 Spring 应用程序上下文管理的对象）。应用程序上下文可以在一个分层结构中配置，在其中，子应用程序上下文可以看到父应用程序上下文中定义的 bean，但反之则不行。默认情况下，Spring 容器在创建容器时验证每个 bean 的配置，包括验证 bean 引用是否实际引用了有效的 bean。对于包含对 SCA 引用和属性引用的 Spring 应用程序上下文，为在 Spring 应用程序上下文中使用的所有 SCA 引用和属性，使得 bean 是 SCA 运行时的职责。然后，Spring 容器可以验证 bean 并成功加载应用程序上下文。

6.9 数据处理框架

随着日益增长的数据量，怎样合理处理这些海量数据是当前研究和应用的热点、重点之一，主要表现就是怎样从这些海量数据中获取、抽取到满足需求的数据。因此，搜索引擎和商业智能迅速得到发展和应用，在搜索引擎最为著名为 Google 和 Baidu 等，商业智能由于集成了数据仓库、联机分析处理和数据挖掘等方法与技术，在处理海量数据方面具有重要的地位。然后经过抽取（Extraction）、转换（Transformation）和装载（Load），即 ETL 过程，合并到一个企业级的数据仓库里，从而得到企业数据的一个全局视图，在此基础上利用合适的查询和分析工具、数据挖掘工具、OLAP 工具等对其进行分析和处理（这时信息变为辅助决策的知识），最后将知识呈现给管理者，为管理者的决策过程提供支持。而本小节主要叙述开源软件搜索框架 Lucene 和数据 ETL 工具 Kettle，并最终在下一章尝试 Lucene 与 Kettle 整合方法。

6.9.1　开源搜索框架 Lucene

Lucene[①]是一套用于全文检索和搜寻的开源程式库，由 Apache 软件基金会支持和提供。Lucene 提供了一个简单确强大的应用程式接口，能够做全文索引和搜寻，在 Java 开发环境里 Lucene 是一个成熟的免费开放源代码工具。Lucene 现已成为最受欢迎的免费 java 资讯检索程式库。人们经常提到资讯检索程式库，就像是搜寻引擎，但是不应该将资讯检索程式库与网搜索引擎相混淆。它最初是由 Doug Cutting 所撰写的，他是一位资深全文索引/检索专家，曾经是 V-Twin 搜索引擎的主要开发者，后来在 Excite 担任高级系统架构设计师，目前从事于一些 INTERNET 底层架构的研究。他贡献出 Lucene 的目标是为各种中小型应用程式加入全文检索功能，于 2001 年 10 月贡献给 APACHE，成为 APACHE 基金 jakarta 的一个子项目。

Lucene 是一个基于 Java 的全文信息检索工具包，它不是一个完整的搜索应用程序，而是为应用程序提供索引和搜索功能。全文检索是指计算机索引程序通过扫描文章中的每一个词，对每一个词建立一个索引，指明该词在文章中出现的次数和位置，当用户查询时，检索程序就根据事先建立的索引进行查找，并将查找的结果反馈给用户的检索方式。这个过程类似于通过字典中的检索字表查字的过程。而全文检索的方法主要分为按字检索和按词检索两种。按字检索是指对于文章中的每一个字都建立索引，检索时将词分解为字的组合。对于各种不同的语言而言，字有不同的含义，比如英文中字与词实际上是合一的，而中文中的字与词有很大分别。按词检索指对文章中的词，即语义单位建立索引，检索时按词检索，并且可以处理同义项等。英文等西方文字由于按照空白切分词，因此实现上与按字处理类似，添加同义处理也很容易。图 6-53 所示是全文检索结构。

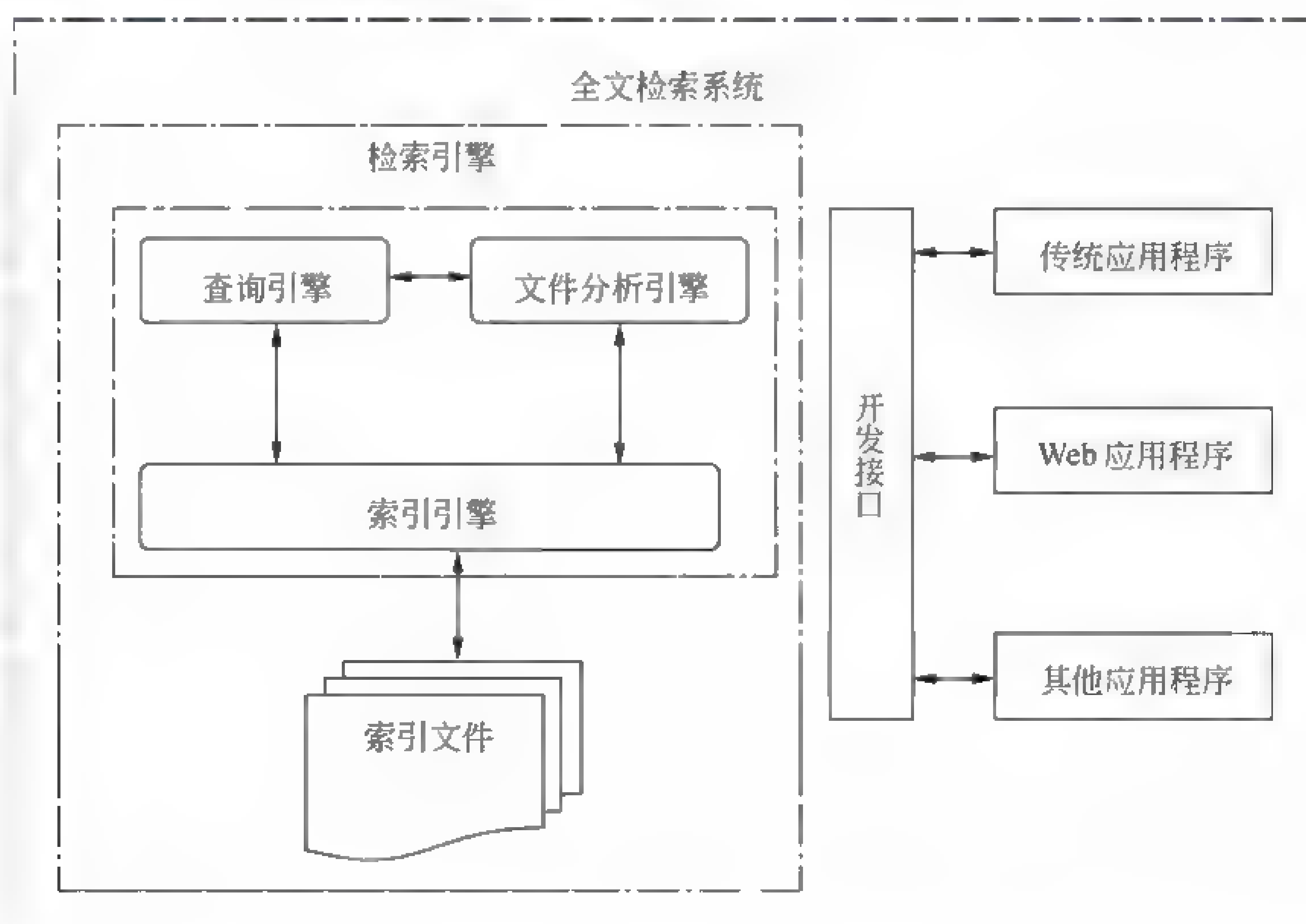

图 6-53　全文检索结构（来源 http://www.lucene.com.cn）

① http://lucene.apache.org

6.9.1.1 Lucene 基本原理

lucene 的检索算法属于索引检索，即用空间来换取时间，对需要检索的文件、字符流进行全文索引，在检索的时候对索引进行快速的检索，得到检索位置，这个位置记录检索词出现的文件路径或者某个关键词。当在使用数据库的项目中，不使用数据库进行检索的原因主要是：数据库在非精确查询的时候使用查询语言“like %keyword%”，对数据库进行查询是对所有记录遍历，并对字段进行“%keyword%”匹配，在数据库的数据庞大，以及某个字段存储的数据量庞大的时候，这种遍历是致命的，它需要对所有的记录进行匹配查询。因此，lucene 主要适用于文档集的全文检索，以及海量数据库的模糊检索，特别是对数据库的 xml 或者大数据的字符类型。

Lucene 能够为文本类型的数据建立索引，所以只要能把要索引的数据格式转化的文本的格式，Lucene 就能对文档进行索引和搜索。比如要对一些 HTML 文档，PDF 文档进行索引的话就首先需要把 HTML 文档和 PDF 文档转化成文本格式，然后将转化后的内容交给 Lucene 进行索引，然后把创建好的索引文件保存到磁盘或者内存中，最后根据用户输入的查询条件在索引文件上进行查询。不指定要索引的文档的格式也使 Lucene 能够几乎适用于所有的搜索应用程序。图 6-54 所示是搜索应用程序和 Lucene 之间的关系。

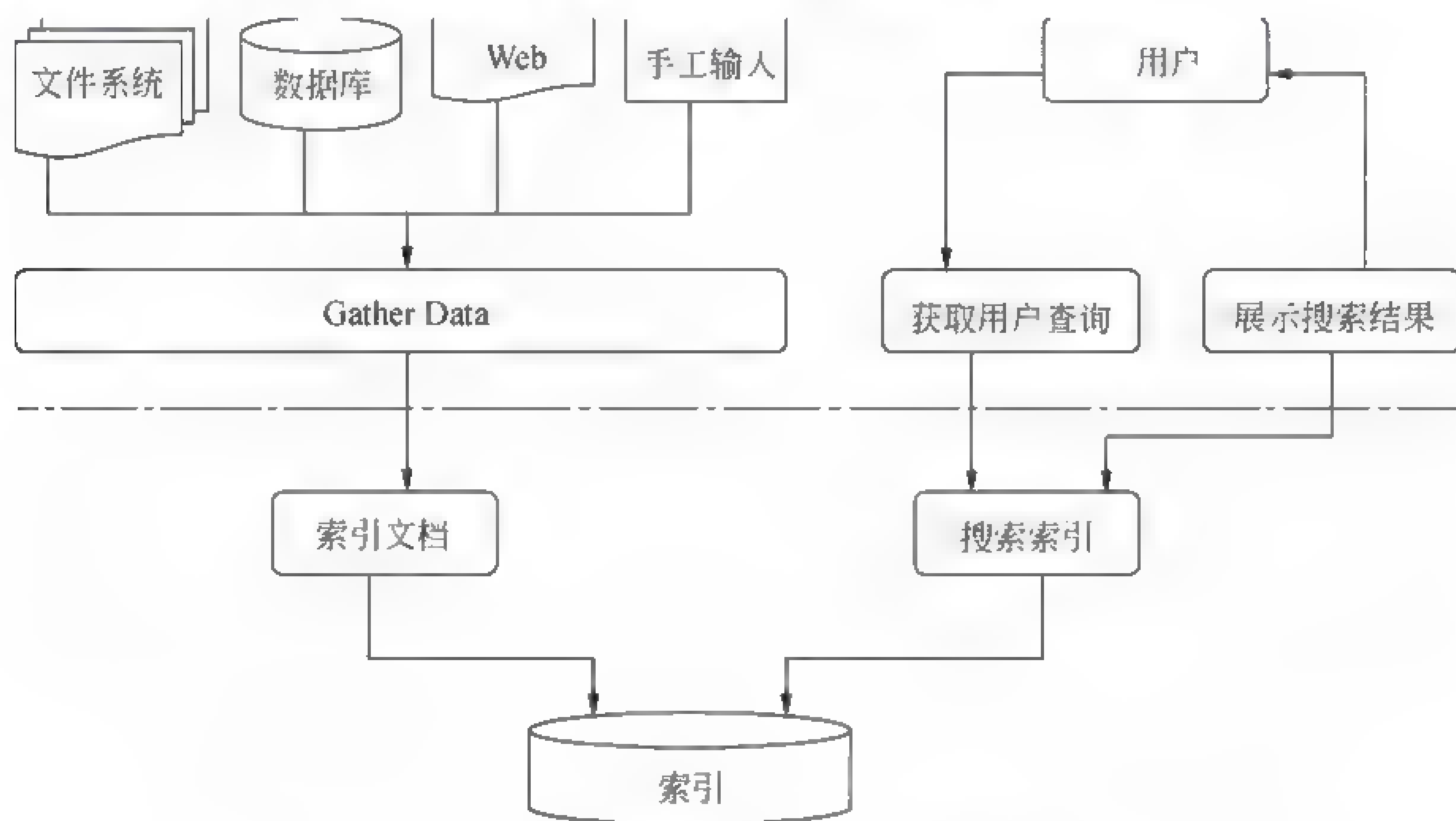

图 6-54 搜索应用程序和 Lucene 之间的关系

Lucene 作为一个优秀的全文检索引擎，其系统结构具有强烈的面向对象特征。首先是定义了一个与平台无关的索引文件格式，其次通过抽象将系统的核心组成部分设计为抽象类，具体的平台实现部分设计为抽象类的实现，此外与具体平台相关的部分比如文件存储也封装为类，经过层层的面向对象式的处理，最终达成了一个低耦合高效率，容易二次开发的检索引擎系统，如图 6-55 所示。

Lucene 的系统由基础结构封装、索引核心、对外接口三大部分组成。其中直接操作索引文件的索引核心又是系统的重点。Lucene 的将所有源码分为了七个模块（在 java 语言中以包即 package 来表示），如表 6-4 所示，各个模块所属的系统部分也如上图所示。需要说明的是

org.apache.lucene.queryPaser 是作为 org.apache.lucene.search 的语法解析器存在，不被系统之外实际调用，因此这里没有当作对外接口看待，而是将之独立出来。从面向对象看，Lucene 应用了最基本的一条程序设计准则：引入额外的抽象层以降低耦合性。首先，引入对索引文件的操作 org.apache.lucene.store 的封装，然后将索引部分的实现建立在（org.apache.lucene.index）其之上，完成对索引核心的抽象。在索引核心的基础上开始设计对外的接口 org.apache.lucene.search 与 org.apache.lucene.analysis。在每一个局部细节上，比如某些常用的数据结构与算法上，Lucene 也充分的应用了这一条准则。在高度的面向对象理论的支撑下，使得 Lucene 的实现容易理解，易于扩展。

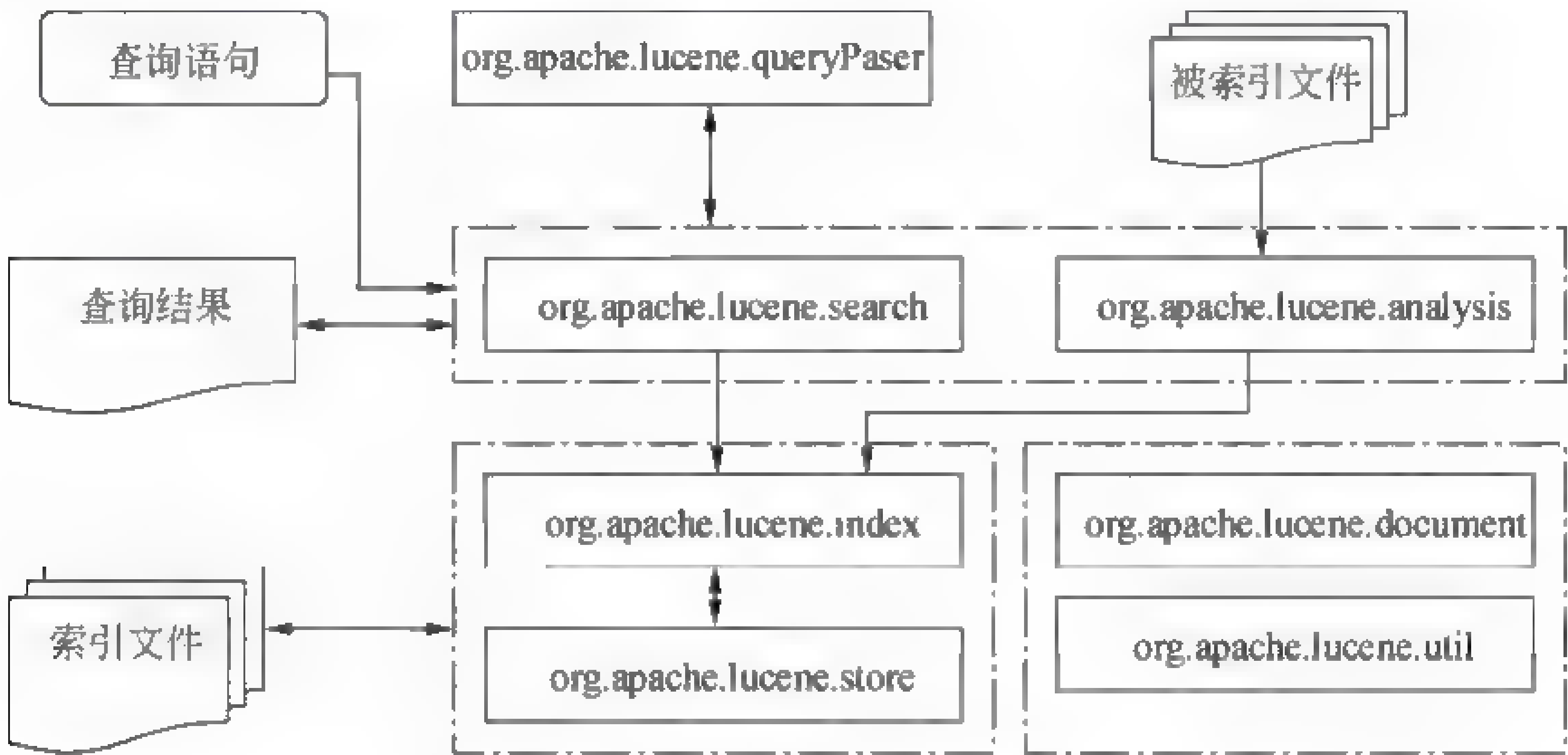

图 6-55　统结构与源码组织图

表 6-4　Lucene 包结构功能表

包名	描述
org.apache.lucene.analysis	语言分析器，主要用于的切词，支持中文主要是扩展此类
org.apache.lucene.document	索引存储时的文档结构管理，类似于关系型数据库的表结构
org.apache.lucene.index	索引管理，包括索引建立、删除等
org.apache.lucene.queryParser	查询分析器，实现查询关键词间的运算，如与、或、非等
org.apache.lucene.search	检索管理，根据查询条件，检索得到结果
org.apache.lucene.store	数据存储管理，主要包括一些底层的 I/O 操作
org.apache.lucene.util	一些公用类

另外，通常将 Solr 构建在 Lucene 功能之上，使 Lucene 可供企业使用，并具有最小的编程需求。因为 Solr 包装并扩展了 Lucene，所以它们使用很多相同的术语。更重要的是 Solr 创建的索引与 Lucene 搜索引擎库完全兼容。通过对 Solr 进行适当的配置，但某些情况下可能需要进行编码，这时 Solr 可以阅读和使用构建到其他 Lucene 应用程序中的索引。在 Solr 和 Lucene 中，使用一个或多个 Document 来构建索引。Document 包括一个或多个 Field。Field 包括名称、内容，以及告诉 Solr 如何处理内容的元数据。例如，Field 可以包含字符串、数字、布尔值或者日期，也可以包含想添加的任何类型。Field 可以使用大量的选项来描述，这些选项告诉 Solr 在索引和搜索期间如何处理内容。如下就是两主要的 Field：

（1）indexed Indexed Field 可以进行搜索和排序，还可以在 indexed Field 上运行 Solr 分析

过程，此过程可修改内容以改进或更改结果。

（2）stored stored Field 内容保存在索引中，这对于检索和醒目显示内容很有用，但对于实际搜索则不是必须的。例如，很多应用程序存储指向内容位置的指针而不是存储实际的文件内容。

同时，在 Solr 和 Lucene 中，Analyzer 包括一个 Tokenizer 和一个或多个 TokenFilter。Tokenizer 负责生成 Token，后者在多数情况下对应要索引的词。TokenFilter 从 Tokenizer 接受 Token，并且可以在索引之前修改或删除 Token。例如，Solr 的 WhitespaceTokenizer 根据空白断词，而它的 StopFilter 从搜索结果中删除公共词。其他类型的分析包括词干提取、同义词扩展和大小写折叠。如果应用程序要求以某种特殊方式进行分析，则 Solr 所拥有的一个或多个断词工具和筛选器可以满足要求。在 Solr 中，通过向部署在 servlet 容器中的 Solr Web 应用程序发送 HTTP 请求来启动索引和搜索。Solr 接受请求，确定要使用的适当 SolrRequestHandler，然后处理请求。通过 HTTP 以同样的方式返回响应，默认配置返回 Solr 的标准 XML 响应，也可以配置 Solr 的备用响应格式。如下程序片段包含了一个 add 命令的例子，当按下 Submit 按钮时向 Solr 发送这个命令：

```
<add>
    <doc>
        <field name="url">http://localhost/myBlog/solr-rocks.html</field>
        <field name="title">Solr Search is Simply Great</field>
        <field name="keywords">solr,lucene,enterprise,search</field>
        <field name="creationDate">2007-01-06T05:04:00.000Z</field>
        <field name="rating">10</field>
        <field name="content">Solr is a really great open source search server.
        It scales,
        it's easy to configure and the Solr community is really
        supportive.</field>
        <field name="published">on</field>
    </doc>
</add>
```

<doc>中的每个 field 条目告诉 Solr 应该将哪些 Field 添加到所创建文档的 Lucene 索引中。可以向 add 命令添加多个<doc>。

6.9.1.2 Lucene 基本应用方法

下面以基于 Lucene 的 Web/用程序为例进行叙述 Lucene 基本应用方法。这是因为 Lucene 能使目前很多服务软件提供商提供搜索功能，也能兼容 Perl、Python、C++和.NET 等流行语言。同时，Lucene 强大的 API 主要关注文本索引和搜索；也可以用于为各种应用程序构建搜索功能，比如电子邮件客户端、邮件列表、Web 搜索、数据库搜索，Wikipedia、TheServerSide、jGuru 和 LinkedIn 等。这些就决定了 Lucene 大体提供如下功能：

（1）拥有强大、准确、有效的搜索算法。

（2）计算每个文档匹配给定查询的分数，并根据分数返回最相关的文档。

（3）支持许多强大的查询类型，比如 PhraseQuery、WildcardQuery、RangeQuery、

FuzzyQuery、BooleanQuery 等。

（4）支持解析人们输入的丰富查询表达式。

（5）允许用户使用定制排序、过滤和查询表达式解析扩展搜索行为。

（6）使用基于文件的锁定机制保护并发索引修改。

（7）允许同时搜索和编制索引。

图 6-56 所示是基于 Lucene 应用程序构建方法。

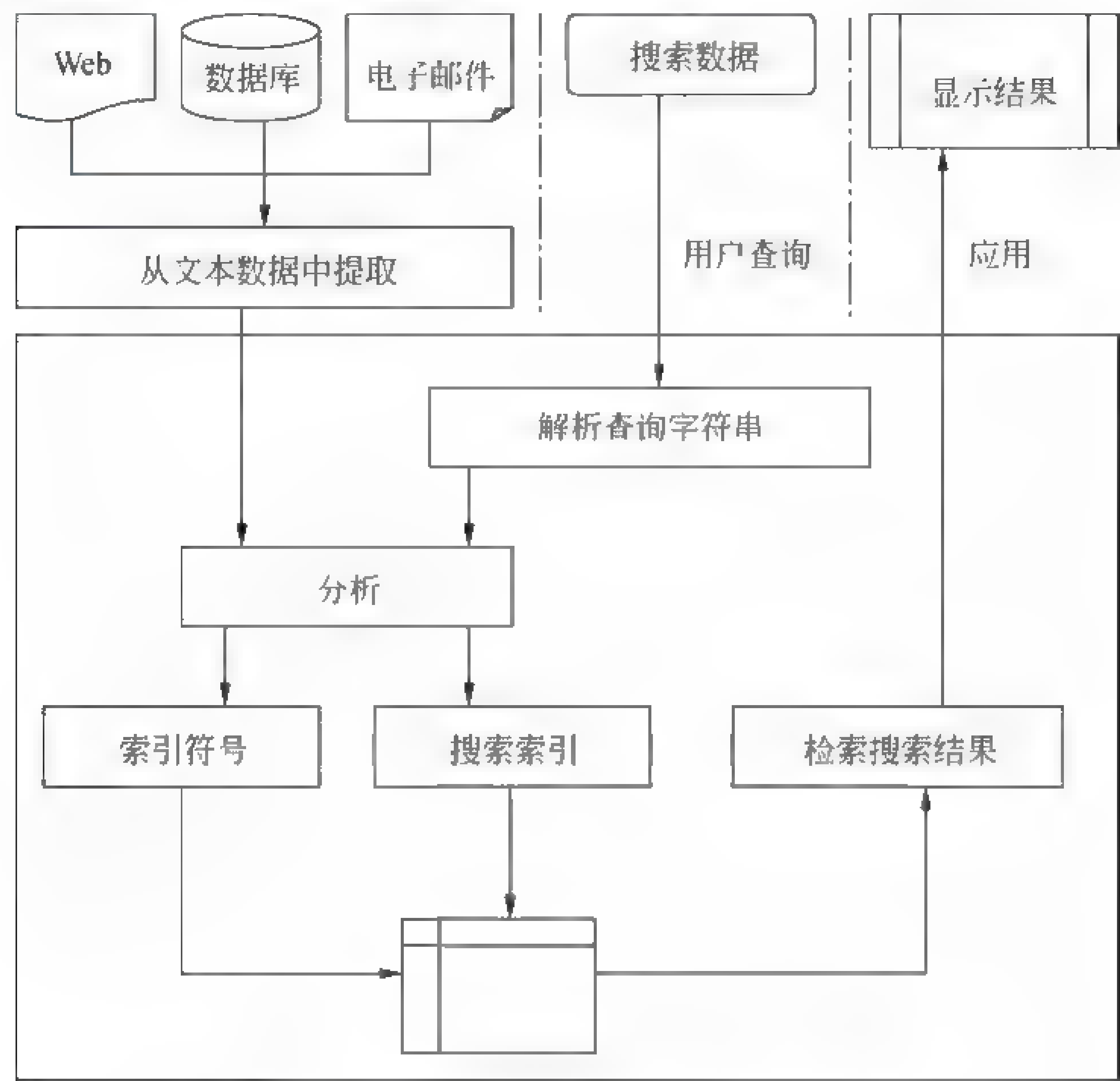

图 6-56　基于 Lucene 应用程序构建步骤

1. Lucene 常用类的分析

编制索引是将文本数据转换为有利于快速搜索的格式，即编制数据索引的第一步是让数据变成一个简单的文本格式。Lucene 将输入数据存储在名为逆序索引的数据结构中，该数据结构以索引文件集的形式存储在文件系统或内存中。大部分 Web 搜索引擎都使用逆序索引，同时允许用户执行快速关键字查询，并将查找匹配给定查询的文档。在将文本数据添加到索引前，由分析程序（使用分析过程）进行处理。而分析是将文本数据转换为搜索基本单位[称为项（term）]的过程。在分析过程中，文本数据将经历多项操作：提取单词、移除通用单词、忽略标点符号、将单词变为词根形式、将单词变成小写等。当分析过程发生在编制索引和查询解析之前，分析将文本数据转换为标记，这些标记将作为项添加到 Lucene 索引中。为此，Lucene 有多种内置分析程序，比如 SimpleAnalyzer、StandardAnalyzer、StopAnalyzer、SnowballAnalyzer 等，它们在标记文本和应用过滤器的方式上有所区别。因为分析在编制索引之前移除单词，它减少了索引的大小，但是不利用精确的查询过程。下面就简要介绍一下核心索引编制类。

（1）Directory：表示索引文件存储位置的抽象类。它有两个常用的子类：FSDirectory：在实际文件系统中存储索引的 Directory 实现，该类对于大型索引非常有用。RAMDirectory：

在内存中存储所有索引的实现。该类适用于较小的索引，可以完整加载到内存中，在应用程序终止之后销毁。由于索引保存在内存中，所以速度相对较快。

（2）Analyzer：分析程序负责处理文本数据并将其转换为标记存储在索引中。使其在编制索引前，IndexWriter 接收用于标记数据的分析程序，要为文本编制索引时，应该使用适用于该文本语言的分析程序。

（3）IndexDeletionPolicy：该接口用来实现从索引目录中定制删除过时提交的策略。默认删除策略是 KeepOnlyLastCommitDeletionPolicy，该策略仅保留最近的提交，并在完成一些提交之后立即移除所有之前的提交。

（4）IndexWriter：创建或维护索引的类，它的构造函数接收布尔值，确定是否创建新索引，或者打开现有索引，它提供在索引中添加、删除和更新文档的方法。如下程序片段就是其使用方法：

```
Directory fsDirectory =  FSDirectory.getDirectory(indexDirectory);
Analyzer standardAnalyzer = new StandardAnalyzer();
boolean create = true;
IndexDeletionPolicy deletionPolicy = new KeepOnlyLastCommitDeletionPolicy();
indexWriter =new IndexWriter(fsDirectory,standardAnalyzer,create,
    deletionPolicy,IndexWriter.MaxFieldLength.UNLIMITED);
```

（5）Searcher：它是一个抽象基类，包含各种超负荷搜索方法。而 IndexSearcher 是 Searcher 的一个常用子类，允许在给定的目录中存储搜索索引。Search 方法返回一个根据计算分数排序的文档集合，Lucene 为每个匹配给定查询的文档计算分数。IndexSearcher 是线程安全的；一个实例可以供多个线程并发使用。

（6）Term：它是搜索的基本单位，由单词文本和出现该文本的字段的名称两部分组成。Term 对象也涉及索引编制，但是可以在 Lucene 内部创建。

（7）Query：它是一个用于查询的抽象基类。搜索指定单词或词组涉及到在项中包装它们，将项添加为查询对象，将查询对象传递到 IndexSearcher 的搜索方法。

Lucene 包含各种类型的具体查询实现，比如 TermQuery、BooleanQuery、PhraseQuery、PrefixQuery、RangeQuery、MultiTermQuery、FilteredQuery、SpanQuery 等。其中：

（1）TermQuery。搜索索引最基本的查询类型，可以使用单个项构建 TermQuery，项值应该区分大小写。传递的搜索项应该与文档分析得到的项一致，这是因为分析程序在构建索引之前对原文本执行许多操作。如下程序片段就是 TermQuery 使用方法：

```
Searcher indexSearcher = new IndexSearcher(indexDirectory);
Term term = new Term("subject"," Lucene ");
Query termQuery = new TermQuery(term);
TopDocs topDocs = indexSearcher.search(termQuery,10);
```

（2）RangeQuery。可以使用 RangeQuery 在某个范围内搜索，索引中的所有项都以字典顺序排列。Lucene 的 RangeQuery 允许用户在某个范围内搜索项，该范围可以使用起始项和最终项（包含两端或不包含两端均可）指定，程序片段如下：

```
Term begin = new Term("date","20090601");
```

```
Term end = new Term("date","20090606");
Query query = new RangeQuery(begin, end, true);
```

（3）PrefixQuery。可以使用 PrefixQuery 通过前缀单词进行搜索，该方法用于构建一个查询，该查询查找包含以指定单词前缀开始的词汇的文档，程序片段如下：

```
PrefixQuery prefixQuery = new PrefixQuery(new Term("sender","job"));
PrefixQuery query = new PrefixQuery(new Term("sender","job"));
```

（4）BooleanQuery。可以使用 BooleanQuery 组合任何数量的查询对象，构建强大的查询，它使用 query 和一个关联查询的子句，指示查询是应该发生、必须发生还是不得发生。在 BooleanQuery 中，子句的最大数量默认限制为 1024。还可以调用 setMaxClauseCount 方法设置最大子句数。程序片段如下：

```
Query query1 = new TermQuery(new Term("subject","java"));
Query query2 = new TermQuery(new Term("subject","bangalore"));
BooleanQuery query = new BooleanQuery();
query.add(query1,BooleanClause.Occur.MUST);
query.add(query2,BooleanClause.Occur.MUST);
```

（5）PhraseQuery。可以使用 PhraseQuery 进行短语搜索，它匹配包含特定单词序列的文档，使用索引中存储项的位置信息。并考虑匹配的项之间的距离称为 slop。默认情况下，slop 的值为零，这可以通过调用 setSlop 方法进行设置。同时，PhraseQuery 还支持多个项短语，程序片段如下：

```
PhraseQuery query = new PhraseQuery();
query.setSlop(1);
query.add(new Term("subject","job"));
query.add(new Term("subject","opening"));
query.add(new Term("subject","j2ee"));
```

（6）WildcardQuery。WildcardQuery 实现通配符搜索查询，这允许搜索 arch*（如可以查找包含 architect、architecture 等）之类的单词。并可以使用两个标准通配符：*（表示零个以上）、?（表示一个以上）。如果使用以通配符查询开始的模式进行搜索，则可能会引起性能的降低，程序片段如下：

```
Query query = new WildcardQuery(new Term("subject","arch*"));
```

（7）FuzzyQuery。可以使用 FuzzyQuery 搜索类似项，该匹配类似于指定单词的单词。类似度测量基于 Levenshtein（编辑距离）算法进行。如下程序片段，FuzzyQuery 用于查找与拼错的单词“admnistrtor”最接近的项，尽管这个错误单词没有索引。

```
subject field. Note we have misspelled admnistrtor here.*/
Query query = new FuzzyQuery(new Term("subject", "admnistrtor"));
```

（8）QueryParser。QueryParser 对于解析人工输入的查询字符非常有用，可以使用它将用

户输入的查询表达式解析为 Lucene 查询对象，这些对象可以传递到 IndexSearcher 的搜索方法。它还可以解析丰富的查询表达式，QueryParser 内部将人们输入的查询字符串转换为一个具体的查询子类。这时需要使用反斜杠（\）将*、?等特殊字符进行转义。并可以使用运算符 AND、OR 和 NOT 构建文本布尔值查询，程序片段如下：

```
QueryParser queryParser = new QueryParser("subject",new StandardAnalyzer());
Query query = queryParser.parse("job openings AND .net AND pune");
```

IndexSearcher：返回一组对分级搜索结果（如匹配给定查询的文档）的引用，并可以使用 IndexSearcher 的搜索方法确定需要检索的最优先搜索结果数量。也可以在此基础上构建定制分页，使其添加定制 Web 应用程序或桌面应用程序来显示搜索结果。检索搜索结果涉及的主要类包括 ScoreDoc（搜索结果中包含一个指向文档的简单指针。这可以封装文档索引中文档的位置以及 Lucene 计算的分数）和 TopDocs（封装搜索结果以及 ScoreDoc 的总数）。

```
TopDocs topDocs = indexSearcher.search(query,20);
System.out.println("Total hits "+topDocs.totalHits);
ScoreDoc[] scoreDosArray = topDocs.scoreDocs;
for(ScoreDoc scoredoc: scoreDosArray){
   Document doc = indexSearcher.doc(scoredoc.doc);
   //Document 是一个字段集合。Lucene 也支持推进文档和字段，这在给某些索引数据赋予重要
     非常有用。
   System.out.println("\nSender: "+dogetField("sender").stringValue());
   System.out.println("Subject: "+dogetField("subject").stringValue());
   System.out.println("Email file location: "
              +doc.getField("emailDoc").stringValue());
}
```

IndexReader：是一个提供各种方法访问索引的抽象类。Lucene 内部引用文档时使用文档编号，该编号可以在向索引添加或从中移除文档时更改，文档编号用于访问索引中的文档。IndexReader 不得用于更新目录中的索引，因为已经打开了 IndexWriter。IndexReader 在打开时总是搜索索引的快照。对索引的任何更改都可以看到，直到再次打开 IndexReader。使用 Lucene 重新打开它们的 IndexReader 可以看到最新的索引更新。如下程序片段表示从索引删除文档为：

```
IndexReader indexReader = IndexReader.open(indexDirectory);
indexReader.deleteDocuments(new Term("month","05"));
indexReader.close();
```

上面，介绍了常用的 Lucene 索引类，下面列举一个较为完整的文档检索程序：

```
package lucene.index;
import java.io.File;
import java.io.FileReader;
import java.io.Reader;
import java.util.Date;
```

```
import org.apache.lucene.analysis.Analyzer;
import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.document.Document;
import org.apache.lucene.document.Field;
import org.apache.lucene.index.IndexWriter;
public class TextFileIndexer {
 public static void main(String[] args) throws Exception{
   File   fileDir  = new File("C:\\files to index ");
   File   indexDir = new File("C:\\luceneIndex");
   Analyzer luceneAnalyzer = new StandardAnalyzer();
   IndexWriter indexWriter = new IndexWriter(indexDir,luceneAnalyzer,true);
   File[] textFiles  = fileDir.listFiles();
   long startTime = new Date().getTime();
   //加入文档索引
   for(int i = 0; i < textFiles.length; i++){
     if(textFiles[i].isFile() >> textFiles[i].getName().endsWith(".txt")){
       System.out.println("File " + textFiles[i].getCanonicalPath()
              + " is being indexed");
       Reader textReader = new FileReader(textFiles[i]);
       Document document = new Document();
       document.add(Field.Text("content",textReader));
       document.add(Field.Text("path",textFiles[i].getPath()));
       indexWriter.addDocument(document);
     }
   }
   indexWriter.optimize();
   indexWriter.close();
   long endTime = new Date().getTime();
   System.out.println("It took " + (endTime - startTime)
              + " milliseconds to create an index for the files in the directory "
              + fileDir.getPath());
  }
}
```

2. Lucene 基本的应用方法

前一节介绍了Lucene常用类的方法，这一节接着介绍Lucene 的 API的其他的一些重要、常用的使用方法。

（1）布尔操作符。大多数的搜索引擎都会提供布尔操作符让用户可以组合查询，典型的布尔操作符有 AND, OR, NOT。Lucene 支持五种布尔操作符，分别是 AND, OR, NOT, 加(+), 减(–)。

```
public void testOperator(String indexDirectory) throws Exception{
   Directory dir = FSDirectory.getDirectory(indexDirectory,false);
   IndexSearcher indexSearcher = new IndexSearcher(dir);
```

```
    String[] searchWords = {"Java AND Lucene", "Java NOT Lucene", "Java OR
    Lucene",
                    "+Java +Lucene", "+Java -Lucene"};
    Analyzer language = new StandardAnalyzer();
    Query query;
    for(int i = 0; i < searchWords.length; i++){
      query = QueryParser.parse(searchWords[i], "title", language);
      Hits results = indexSearcher.search(query);
      System.out.println(results.length() + "search results for query " +
      searchWords[i]);
    }
}
```

（2）域搜索（Field Search）。Lucene 支持域搜索，使得可以指定一次查询是在哪些域（Field）上进行。例如，如果索引的文档包含两个域，Title 和 Content，就可以使用查询“Title: Lucene AND Content: Java”来返回所有在 Title 域上所包含的 Lucene 文档，并且在 Content 域上包含 Java 的文档：

```
public void testFieldSearch(String indexDirectory) throws Exception{
    Directory dir = FSDirectory.getDirectory(indexDirectory,false);
    IndexSearcher indexSearcher = new IndexSearcher(dir);
    String searchWords = "title:Lucene AND content:Java";
    Analyzer language = new StandardAnalyzer();
    Query query = QueryParser.parse(searchWords, "title", language);
    Hits results = indexSearcher.search(query);
    System.out.println(results.length() + "search results for query " +
    searchWords);
}
```

（3）通配符搜索（Wildcard Search）。Lucene 支持两种通配符：问号（？）和星号（*），可以使用问号（？）来进行单字符的通配符查询，或者利用星号（*）进行多字符的通配符查询。程序片段如下：

```
public void testWildcardSearch(String indexDirectory)throws Exception{
    Directory dir = FSDirectory.getDirectory(indexDirectory,false);
    IndexSearcher indexSearcher = new IndexSearcher(dir);
    String[] searchWords = {"tex*", "tex?", "?ex*"};
    Query query;
    for(int i = 0; i < searchWords.length; i++){
      query = new WildcardQuery(new Term("title",searchWords[i]));
      Hits results = indexSearcher.search(query);
      System.out.println(results.length() + "search results for query " +
      searchWords[i]);
    }
}
```

（4）模糊查询。Lucene 提供的模糊查询是基于编辑距离算法（Edit distance algorithm）来实现的。可以在搜索词的尾部加上字符～来进行模糊查询。例如，查询语句“think~”返回所有包含和 think 类似的关键词的文档，程序片段如下：

```
public void testFuzzySearch(String indexDirectory)throws Exception{
   Directory dir = FSDirectory.getDirectory(indexDirectory,false);
   IndexSearcher indexSearcher = new IndexSearcher(dir);
   String[] searchWords = {"text", "funny"};
   Query query;
   for(int i = 0; i < searchWords.length; i++){
      query = new FuzzyQuery(new Term("title",searchWords[i]));
      Hits results = indexSearcher.search(query);
      System.out.println(results.length() + "search results for query " +
      searchWords[i]);
   }
}
```

（5）范围搜索（Range Search）。范围搜索匹配是在某域上搜索一定范围的文档。例如，查询“age:[18 TO 35]”返回所有 age 域上的值在 18～5 之间的文档。示例如下：

```
public void testRangeSearch(String indexDirectory)throws Exception{
    Directory dir = FSDirectory.getDirectory(indexDirectory,false);
    IndexSearcher indexSearcher = new IndexSearcher(dir);
    Term begin = new Term("birthDay","20000101");
    Term end   = new Term("birthDay","20110910");
    Query query = new RangeQuery(begin,end,true);
    Hits results = indexSearcher.search(query);
    System.out.println(results.length() + "search results is returned");
}
```

如下程序片段代码就是一个基于 Lucene 的搜索功能的实现方法：

```
package sample.dw.paper.lucene.search;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import org.apache.lucene.analysis.Analyzer;
import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.queryParser.ParseException;
import org.apache.lucene.queryParser.QueryParser;
import org.apache.lucene.search.Hits;
import org.apache.lucene.search.IndexSearcher;
import org.apache.lucene.search.Query;
import sample.dw.paper.lucene.index.IndexManager;
public class SearchManager {
    private String searchWord;
```

```
    private IndexManager indexManager;
    private Analyzer analyzer;
    public SearchManager(String searchWord){
        this.searchWord   =  searchWord;
        this.indexManager =  new IndexManager();
        this.analyzer     =  new StandardAnalyzer();
    }
    public List search(){
        List searchResult = new ArrayList();
        if(false == indexManager.ifIndexExist()){
        try {
            if(false == indexManager.createIndex()){
                return searchResult;
            }
        } catch (IOException e) {
          e.printStackTrace();
          return searchResult;
        }
        }
     IndexSearcher indexSearcher = null;
        try{
            indexSearcher = new IndexSearcher(indexManager.getIndexDir());
        }catch(IOException ioe){
            ioe.printStackTrace();
        }
        QueryParser queryParser = new QueryParser("content",analyzer);
        Query query = null;
        try {
            query = queryParser.parse(searchWord);
        } catch (ParseException e) {
          e.printStackTrace();
        }
        if(null != query >> null != indexSearcher){
            try {
                Hits hits = indexSearcher.search(query);
                for(int i = 0; i < hits.length(); i ++){
                    SearchResultBean resultBean = new SearchResultBean();
                    resultBean.setHtmlPath(hits.doc(i).get("path"));
                    resultBean.setHtmlTitle(hits.doc(i).get("title"));
                    searchResult.add(resultBean);
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
```

```
            return searchResult;
        }
    }
```

在程序片段的类里面有三个私有属性，第一个是 searchWord，代表了来自客户端的搜索词。第二个是 indexManager，代表了在索引子系统中定义的类 IndexManager 的一个实例。第三个是 analyzer，代表了用来解析搜索词的解析器。

这个方法首先检查索引文件是否已经存在，如果已经存在，那么就在已经存在的索引上进行检索，如果不存在，那么首先调用类 IndexManager 提供的方法来创建索引，然后在新创建的索引上进行检索。搜索结果返回后，这个方法从搜索结果中提取出需要的属性，并为每个搜索结果生成类 SearchResultBean 的一个实例。最后这些 SearchResultBean 的实例被放到一个列表里面，并返回给请求管理器。在类 SearchResultBean 中，含有两个属性，分别是 htmlPath 和 htmlTitle，以及这两个属性的 get 和 set 方法。这也意味着我们的搜索结果包含两个属性：htmlPath 和 htmlTitle，其中 htmlPath 代表了 HTML 文件的路径，htmlTitle 代表了 HTML 文件的标题。

6.9.2　多源数据抽取框架

世界各地的企业需要在应用程序之间频繁地转移大量数据，这些数据转移的共同点是希望让人们更好地查看和控制数据，这还不仅仅是应用程序之间的数据转移，数据仓库、主数据管理、分析和业务智能化都需要完成大量数据集成任务，从而执行批处理、微型批处理和实时处理。其中数据抽取是 ETL 工作的第一步，也就是从多源数据中获取数据的重要一步，这些数据源主要包括结构化的关系数据库的数据，半结构化的 XML 数据及非结构化的语义数据。为了解决数据集成问题，引入了提取、转换和装载（Extract Transform and Load，ETL）工具来有效解决这个问题。

6.9.2.1　ETL

目前，XML 已经广泛应用于各种类型软件中。在软件配置、数据交换、WEB 服务、云计算等我们都可以找到 XML 的应用，可以说 XML 无处不在。因此在很多企业的数据整合项目中，都需要考虑如何整合 XML 数据。这也逐渐使得 XML 的数据量大，难以获取到满足需求的数据。这些数据主要表现在：合并、收购、全球化、竞争的压力和各种其他因紧迫使企业有效地利用他们的信息，从中挖掘出更大的业务价值。但是，这些业务情形常常会产生许多与信息相关的难题。它们涉及大量数据，常常对数据的内容、质量和结构缺乏足够的了解。业务数据包括来自客户和合作伙伴的复杂的业务事务，以及在企业内部流动的各种操作信息，常常必须根据这些信息做出关键的业务决策。这时，通常将从多个源应用程序（内部的和外部的）收集事务数据称作抽取（Extract）、转换（Transform）和装入（Load），缩写为 ETL。首先从无数的应用程序或仓库中抽取数据。然后，普遍地将之转换为符合且便于在数据仓库中进行分析的模式。其中，数据仓库是为高性能的复杂查询而设计的关系数据库。一旦将数据装入到数据仓库或数据集市中，该数据仓库就成为了一站式商店（one stop shop），用于预测库存、开销和收入；一般对于任何情况，都可用历史性能来预测趋势。这里重要的是要注

意数据集市与数据仓库之间的区别。大多数数据集市是特定于应用程序的，致力于解决一组特殊的业务问题，而数据仓库主要是为了集结所有业务领域的信息。图 6-57 所示是 ETL 实现过程。

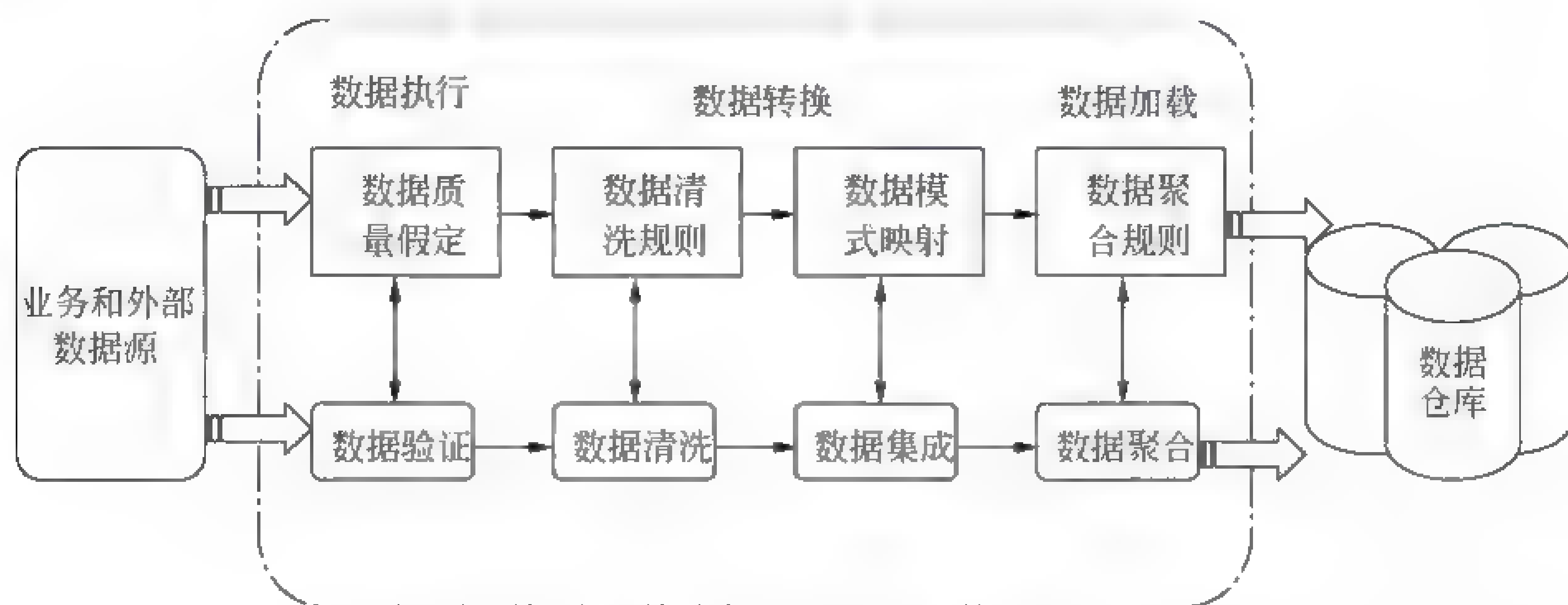

图 6-57　ETL 过程

ETL 过程是将原始数据从业务数据库或其他数据源进行抽取，转换并最终加载到用于分析的数据仓库模型中的过程。对于一般的数据仓库系统，通常需要进行 ETL 转换，并需要将来自于不同数据源的原始数据进行清洗，转换和聚合，将它们转换成易于进行分析的数据仓库数据。具体来讲，ETL 的不同阶段指：

（1）抽取（Extract）是将数据从源数据系统抽取到目标数据仓库中，通常抽取可能会涉及到从多个源数据系统中提取数据。

（2）转换（Transform）是将已经抽取的数据放到数据仓库中，并根据一系列或者多个层次的规则进行转换，使它成为数据仓库模型能够接受的模式。

（3）加载（Load）是将转换后的数据最终加载到用于分析的数据模型中。

下面叙述与 ETL 相关的数据集市、主数据和元数据的基本概念。

1．数据集市

数据集市又称数据市场，是一个从操作的数据和其他的为某个特殊的专业人员团体服务的数据源中收集数据的仓库。从范围上来说，数据是从企业范围的数据库、数据仓库，或者是更加专业的数据仓库中抽取出来的。数据中心的重点就在于它迎合了专业用户群体的特殊需求，在分析、内容、表现，以及易用方面。

数据集市将合并不同系统的数据源来满足业务信息需求。若能有效地得以实现，数据集市将可以快速且方便地访问简单信息，以及系统的和历史的视图。一个设计良好的数据集市将：

（1）发布特定用户群体所需的信息，且无需受制于源系统的大量需求和操作性危机。

（2）支持访问非易变（nonvolatile）的业务信息。（非易变的信息是以预定的时间间隔进行更新的，并且不受 OLTP （On Line Transaction Processing）系统进行中的更新的影响。）

（3）调和来自于组织里多个运行系统的信息，比如账目、销售、库存和客户管理以及组织外部的行业数据。

（4）通过默认有效值、使各系统的值保持一致，以及添加描述以使隐含代码有意义，从而提供净化的（cleansed）数据。

（5）为即席分析和预定义报表提供合理的查询响应时间（不同于 OLTP 系统中所需的调优需求）。

（6）通过提供对于遗留系统和 OLTP 应用程序的选择来减少对这些应用程序的要求，以获得更多所需信息。

2．主数据

企业主数据是用来描述企业核心业务实体的数据，比如客户、合作伙伴、员工、产品、物料单、账户等；它是具有高业务价值的、可以在企业内跨越各个业务部门被重复使用的数据，并且存在于多个异构的应用系统中。还可以包括很多方面，除了常见的客户主数据之外，不同行业的客户还可能拥有其他各种类型的主数据，例如：对于电信行业客户而言，电信运营商提供的各种服务可以形成其产品主数据；对于航空业客户而言，航线、航班是其企业主数据的一种。对于某一个企业的不同业务部门，其主数据也不同，例如市场销售部门关心客户信息，产品研发部门关心产品编号、产品分类等产品信息，人事部门关心员工机构，部门层次关系等信息。然而，主数据则定义企业核心业务对象，如客户、产品、地址等，与交易流水信息不同，主数据一旦被记录到数据库中，需要经常对其进行维护，从而确保其时效性和准确性；主数据还包括关系数据，用以描述主数据之间的关系，如客户与产品的关系、产品与地域的关系、客户与客户的关系、产品与产品的关系等。主数据管理是指一整套的用于生成和维护企业主数据的规范、技术和方案，以保证主数据的完整性、一致性和准确性。

集成、共享、数据质量、数据治理是主数据管理的四大要素，主数据管理要做的就是从企业的多个业务系统中整合最核心的、最需要共享的数据（主数据），集中进行数据的清洗和丰富，并且以服务的方式把统一的、完整的、准确的、具有权威性的主数据分发给全企业范围内需要使用这些数据的操作型应用和分析型应用，这包括各个业务系统、业务流程和决策支持系统等。主数据管理使得企业能够集中化管理数据，在分散的系统间保证主数据的一致性，改进数据合规性、快速部署新应用、充分了解客户、加速推出新产品的速度。从 IT 建设的角度，主数据管理可以增强 IT 结构的灵活性，构建覆盖整个企业范围内的数据管理基础和相应规范，并且更灵活地适应企业业务需求的变化。

在一个完整的主数据管理解决方案中，除了主数据管理的核心服务组件之外通常还会涉及到企业元数据管理、企业信息集成、ETL、数据分析和数据仓库，以及 EAI/ESB 等其他各种技术和服务组件。其中主数据管理服务又包括如下一些主要的服务组件：

（1）Interface Services：为企业中需要主数据的所有业务系统提供各种服务接口，通过实时的、批量的接口可以读取或者修改主数据，这些接口包括 Batch, Web Services, XML Interface, Messaging Interface, Publish/Subscribe, Import/Export Services, Data Standardization Interface, Directory Integration 等。

（2）Lifecycle Management Services：履行针对主数据的 CRUD 操作，执行对主数据存储库中的数据进行更新、存取和管理时的业务逻辑，除此之外，它还负责维护主数据的衍生信息，例如客户之间的关系、客户的偏好、客户在各种客户服务渠道上的行为轨迹等。Lifecycle Management Services 贯穿整个主数据管理的生命周期，它利用 Data Quality Management Services 来确保数据质量、利用 Master Data Event Management Services 来捕获各种主数据变化等相关的事件，以及利用 Hierarchy and Relationship Management Services 用来维护数据实体之间的关系和层次。

（3）Data Quality Management Services：确保主数据的质量和标准化，这在主数据管理解决方案中是一个非常重要的组件，并在各个业务系统获取数据之后，要对数据进行清洗和验证，例如对于地址而言，要弥补地址的缺失、地市的缺失、邮编的缺失、进行地址的标准化等。对于其他数据要进行非空检查、外键检查、数据过滤等。然后要对数据进行匹配/重复识别、自动进行基于规则的合并/去重、交叉验证等，并且还要遵从企业的数据管控规范和流程。它可以是 Master Data Management Services 的一个内部组件，也可以调用整个企业的 Information Integrity Services 来实现。

（4）Authoring Services：依据数据管控流程，定义和扩展企业的主数据模型。

（5）Hierarchy Relationship and Management Services：定义数据实体的层次（Hierarchy）、分组（Grouping）、关系（Relationship）和版本（Version）等。

（6）Master Data Event Management Services：捕获事件并且触发相应的操作，包括事件发现、事件管理和通知功能，它在主数据管理系统和业务系统之间进行数据同步时起到至关重要的作用。

（7）Base Services：提供通用服务，包括安全控制、错误处理、交易日志、事件日志等功能。

（8）Master Data Repository：主数据存储库，包括 Metadata、Master Data、History Data、Reference Data 等。

3．元数据

元数据（Metadata）通常被称为关于数据的数据，是从数据仓库活动中产生的另一功能。在数据集市中，用户将在成百上千个数据元素中进行选择，而这些数据元素是来自于多个系统的，用户还将需要很好地理解这些数据元素以回答所提出的商业问题。元数据将通过提供数据定义、转换逻辑、有效值列表、业务逻辑等来支持这一信息需求。元数据软件的主要组件包括存储所有信息的仓库、用户界面、与其他软件的接口，以及电子和纸张发布组件。

元数据可以描述信息系统的设计、开发和实现，还可以描述数据的流动。比如像数据库表的描述、数据库对象之间的关系、表中列的定义、表中列的数据类型和转换列的数据派生规则等可以认为是元数据。元数据在分类上可以分为业务元数据、技术元数据和操作元数据三类。表 6-5 所示是这三类元数据对照表。

表 6-5　业务元数据、技术元数据和操作元数据对照表

	描述	示例	使用者
业务元数据	业务元数据对于为集成项目提供上下文非常重要。它帮助用日常语言定义词汇，与技术实现无关	使用业务语言的业务规则、负责人、业务定义、审计词汇、词汇表和算法	业务用户
技术元数据	主要由技术人员使用，比如开发人员。这包括表定义和数据类型等内容。在应用程序设计和开发过程中常常使用这些对象	源和目标系统的定义、表和字段的结构和属性、用于审计派生和依赖关系的文档	特定工具的用户如 BI、ETL、剖析、建模
操作元数据	是指在执行过程时生成和捕捉的元数据。它让管理员可以管理系统，确保系统运行顺畅。操作元数据还有助于管理员排除过程中发生的问题	关于应用程序运行的信息，包括频率、记录数量、对每个组件的分析以及用于审计的其他统计数据	操作、管理和业务用户

6.9.2.2　开源框架 Kettle

Kettle 是一款国外开源的 ETL 工具，由开源协议 LGPL 授权，纯 Java 编写，绿色无须安装，且数据抽取高效稳定（数据迁移工具）。Kettle 中有两种脚本文件、transformation 和 Job，transformation 完成针对数据的基础转换，Job 则完成整个工作流的控制。Kettle 这个 ETL 工具集，它允许管理来自不同数据库的数据，通过提供一个图形化的用户环境来描述，它是商业智能开源软件 Pentaho 的一个重要组成部分，现在在国内项目应用上逐渐增多。图 5-58 是 Kettle 的结构图。

图 6-58　Kettle 结构图

整个 Kettle 由 Spoon、Pan、Kitchen 和 Carte 等四部分组成，Spoon 是一个图形用户界面，它允许运行转换或者任务，其中转换是用 Pan 工具来完成，任务是用 Kitchen 来完成。其中 Pan 是一个数据转换引擎，它可以执行很多功能，例如：从不同的数据源读取、操作和写入数据。Kitchen 是一个可以运行利用 XML 或数据资源库描述的任务。通常任务是在规定的时间间隔内用批处理的模式自动运行。现对 Kettle 中的转换和任务的解释对照表格分别见表 6-6、表 6-7 所示。

表 6-6　Kettle 中的转换

名称	描述
Value	Values 是行的一部分，并且是包含以下类型的的数据：Strings、floating point Numbers、unlimited precision BigNumbers、Integers、Dates、或者 Boolean
Row	一行包含 0 个或者多个 Values
Output Stream	一个 Output Stream 是离开一个步骤时的行的堆栈
Input Stream	一个 Input Stream 是进入一个步骤时的行的堆栈
Hop	一个 Hop 代表两个步骤之间的一个或者多个数据流，一个 Hop 总是代表着一个步骤的输出流和一个步骤的输入流
Note	一个 Note 是一个转换附加的文本注释信息

表 6-7 Kettle 中的转换任务

名称	描述
Job Entry	一个 Job Entry 是一个任务的一部分，它执行某些内容
Hop	一个 Hop 代表两个步骤之间的一个或者多个数据流。一个 Hop 总是代表着两个 JobEntry 之间的连接，并且能够被原始的 Job Entry 设置，无条件的执行下一个 Job Entry，直到执行成功或者失败
Note	一个 Note 是一个任务附加的文本注释信息

小 结

本章详细叙述、分析、概述了目前常用的开源软件框架，这些开源软件包括解决 AJAX 直接调用 Bean 的 DWR，门户框架 Liferay、Jetspeed；轻量级开源软件 SSH（Struts、Spring、Hibernate/iBatis）；Web 服务开源框架 CXF、Axis、Tuscany，包括语义 Web 服务转换工具 WSDL2OWL、建模工具 Protégé；全文检索开源框架 Lucene 和 ETL 开源框架 Kettle 等。对这些开源软件框架从易学习、易掌握、易操作等角度进行了全面的总结和分析，并浓缩了这些开源软件原理、技术的精华，这能让读者快速入门、快速掌握和快速的学习。而且这些开源软件基本上都是具有成熟性、可操作性和可用性等特征，甚至有些开源框架已经在相关业务领域等到了很好的应用，也创造了巨大的经济和社会效益。

特别地，对这些开源软件框架进行了分类、整合和比较，也分析指出了这些开源软件框架的基本技术结构和应用方法，并举例进行了分析和说明。

第 7 章　多开源软件框架整合方法

通常需要根据业务需求将不同的开源软件整合集成在 Eclipse 中实现软件系统，这是开源软件应用灵活性直接表现之一，也是根据不同的业务需求选择不同的开源软件来提高软件研发效率和软件复用率。

7.1　概述

本章主要内容是将本书所涉及到的开源软件进行有效整合，即将本书可以整合开源软件进行有效融合，其实整合就是对基于 XML 的配置文件进行配置，但这种配置需要得到各方开源软件的支持和兼容，且各开源软件间具有互访的接口，最终在后面的两章将对此整合进行应用。当然有一些开源软件不需要整合，同样可以集成在 Eclipse 中进行应用研发，如将 Lucene 和 ETL 开源框架 Kettle 集成在 Eclipse 分别根据不同的需求开发出不同的功能。

7.2　PP：面向 AJAX 的 DWR 与 Jetspeed 整合

要实现 PP 的整合，首先是实现 AJAX 与 DWR 的整合，其通过在 Web.xml 中配置 servlet，DWR 使用这些 servlet 来实现接受和响应 AJAX 请求，并在 DWR 中配置 dwr.xml 来告诉 DWR 远程使用 Java 类（JaveBean）。然后通过 DWR 实现 AJAX 与 portlet（Jetspeed）通信，这样就完成了 PP 页面处理整合，程序代码如下：

7.2.1　配置 web.xml 格式

```
...
<servlet>
  <servlet-name>dwr-invoker</servlet-name>
  <display-name>DWR Servlet</display-name>
  <servlet-class>uk.ltd.getahead.dwr.DWRServlet</servlet-class>
  <init-param>
    <param-name>debug</param-name>
    <param-value>true</param-value>
  </init-param>
 </servlet>
 <servlet mapping>
  <servlet name>dwr invoker</servlet name>
  <url pattern>/dwr/*</url-pattern>
```

```
 </servlet mapping>
...
```

7.2.2 配置 dwr.xml 格式

```
<!DOCTYPE dwr PUBLIC
    "-//GetAhead Limited//DTD Direct Web Remoting 1.0//EN"
    "http://www.getahead.ltd.uk/dwr/dwr10.dtd">
 <dwr>
  <allow>
    <create creator="new" javascript="dwr 库">
//creator 属性用来指定使用哪种创造器。如果 creator 属性被设置为值 new，意味着 DWR 调
  用类的默认构造函数来获得实例
      <param name="class" value="类名"/>
    </create>
...
  </allow>
 </dwr>
```

7.2.3 配置 portlet.xml 格式

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app
    xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
    version="1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app 1 0
    .xsd http://java.sun.com/xml/ns/portlet/portlet-app 1 0.xsd"
    id="InterPortletMessagingUsingAJAX.81af06ffa0">
    <portlet>
        <portlet-name> portlet 名</portlet-name>
        <display-name> portlet 名</display-name>
        <display-name xml:lang="en"> portlet 名</display-name>
        <portlet-class>
            interportletmessagingusingajax. portlet 名
       /* DWR 实现 AJAX 与 portlet（Jetspeed）通信*/
        </portlet-class>
        <init-param>
            <name>wps.markup</name>
            <value>html</value>
        </init-param>
        <expiration-cache>0</expiration-cache>
        <supports>
            <mime-type>text/html</mime type>
```

```
            <portlet mode>view</portlet mode>
        </supports>
        <supported locale>en</supported locale>
        <resource-bundle>
            interportletmessagingusingajax.nl.OrdersResource
        </resource-bundle>
        <portlet-info>
            <title>Orders</title>
        </portlet-info>
    </portlet>
    ...
</portlet-app>
```

同时，在 Spring MVC 中也可以使用 DWR，使用方法配置如下：

（1）配置 DWR 的格式。

```
<dwr>
 <allow>
 <create creator="spring" javascript="testname " scope="application">
 <param name="beanName" value="  "/>
 <include method=" "/>
 ...
 </create>
 <convert converter="bean" match="com.test.*">
 <param name="include" value="name"/>
 </convert>
 </allow>
 </dwr>
```

在配置代码中，creator="spring"提供了一个 Spring 的创造器，允许直接调用 Spring 容器中的 bean，然后 DWR 将 bean 转换成一个 javascript 对象。SpringCreator 这个创造器会查找 Spring 所配置的 bean，并创建它们。Javascript 属性用于指定浏览器中被创造出来的对象的名字。scope 属性指定这个 bean 的生命周期，以及嵌套在 create 元素内的 param 元素的 name 属性值可以是 class，beanName 等。此处用 beanName，value 的值 Javabean 是在 beans.xml 中定义的某个 id 值。include 元素指定公开的方法名称，也可以用 exclude 元素指定不想被访问的方法。convert 元素的作用是告诉 DWR 在服务器端的 Java 对象表示和 JavaScript 之间如何转换数据类型。同时，DWR 能自动地在 Java 对象和 JavaScript 表示之间转换简单数据类型，这些类型包括 Java 原生类型和它们各自的类表示，还有数组和集合类型。

（2）配置 beans.xml 的格式。

```
<beans>
   <bean id="testname" class="*.service.impl. testname Impl"/>
 </beans>
```

（3）配置 web.xml 的格式。

```
...
```

```
<context param>
 <param name>contextConfigLocation</param name>
 <param value>/WEB INF/beans.xml</param value>
/*找配置文件 beans.xml */
 </context-param>
 <listener>
 <listener-class>
 org.springframework.web.context.ContextLoaderListener
 </listener-class>
 </listener>
...
<init-param>
 <param-name>debug</param-name>
 <param-value>true</param-value>
 </init-param>
... ...
<init-param>
    <param-name>classes</param-name>
    <param-value>
        com.osl.service.impl.UserManagerImpl,
        com.osl.entity.User,
    </param-value>
 </init-param>
/*使用 DWR 标注*/
```

（4）在 JSP 页面中使用 DWR。

```
<script type='text/javascript' src='dwr/interface/testname.js'></script>
 <script type='text/javascript' src='dwr/engine.js'></script>
 <script type='text/javascript' src='dwr/util.js'></script>
```

（5）在 mvc-config.xml 里配置 controller。

```
<bean id="registerController" class=" ">
    <property name="commandClass" value=" ">
 </property>
 <property name="testname" ref=" testname "></property>
 <property name="successView" value="account/success"/>
 <property name="pages">
 <list>
 <value>test</value>
 <value>test</value>
 </list>
 </property>
 </bean>
```

7.3　SSH 整合

SSH 整合指的是 Struts、Spring、Hibernate 三种经典开源软件框架的集成，以及 Struts、Spring 与 iBatis 的整合，还包括 Struts 与 PP 的融合。当然，这些开源软件框架的整合也主要表现在基于 XML 的配置文件上。

7.3.1　概述

基于 SSH 的轻量级开源软件框架目前已在诸多领域得到了很好的应用，也帮助了相关软件企业提高了软件的研发效率。这种该框架不但具有各种轻量级开源件的优点，而且还具有集成性，即可以通过相关的适配器和 XML 数据总线集成遗留系统和新系统。也能有效解决目前现存的相关的信息化系统都面临很多缺点，如集成性不强、可扩展弱、开发困难、生命周期短等，但出现这些缺点很大的因素是受所选择构架方式相关的，因此，在 J2EE 框架支持下采用 SSH 集成框架来进行构建，以来增强所面临的相关缺点。

目前，SSH 也是最为 Java 业界熟知的 Java EE Web 组件层的开发技术，相关的书籍、教程等都以该整合框架为例进行讲解和应用，这对普及和推动 SSH 的应用起到了极大的推动作用。但大多都是直接讲解各种标签、配置的用法，这给许多初级读者带了极大的困难，毕竟众多初学者对 XML 等相关技术未能很好的熟悉。针对此，本书对 SSH 的整合，分步骤、分层次、从细节角度进行配置说明，再加之相关读者学习本书前面的内容，就很容易掌握，并进行创新，从而增强自身在软件研发的能力。

7.3.2　Struts 与 Spring 整合

与 Struts 相似，Spring 可以作为一个 MVC 实现。由于利用 Struts 可以作为构造高品质软件的基础，以至于开发团队宁愿整合 Spring 框架的特性，而不愿意转换成 Spring MVC。而且 Spring 架构允许将 Struts 作为 Web 框架连接到基于 Spring 的业务和持久层。当前共有三种基本方法可以实现 Struts 与 Spring 整合：

（1）使用 Spring 的 ActionSupport 类整合 Structs；

（2）使用 Spring 的 DelegatingRequestProcessor 覆盖 Struts 的 RequestProcessor；

（3）将 Struts Action 管理委托给 Spring 框架。

但这三种方法都需要使用 Spring 的 ContextLoaderPlugin 为 Struts 的 ActionServlet 装载 Spring 应用程序环境，就像添加任何其他插件一样，则在 struts-config.xml 中配置如下：

```
<plug-in className=
  "org.springframework.web.struts.ContextLoaderPlugIn">
    <set-property property=
      "contextConfigLocation" value="/WEB-INF/beans.xml"/>
 </plug-in>
```

7.3.2.1 使用 Spring 的 ActionSupport 类整合 Structs

这是一种整合 Struts 和 Spring 的最直观的方式，为了方便地获得 Spring 环境，org.springframework.web.struts.ActionSupport 类提供了一个 getWebApplicationContext() 方法。只是采用 Spring 的 ActionSupport 而不是 Struts Action 类扩展动作，它是将 Struts 动作与 Spring 框架耦合在一起。如果需要替换掉 Spring，那么需要必须重写代码。由于 Struts 动作不在 Spring 的控制之下，所以它也不能获得 Spring AOP 的优势。如下程序片段就是使用 ActionSupport 整合 Struts：

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.DynaActionForm;
import org.springframework.context.ApplicationContext;
import org.springframework.web.struts.ActionSupport;
public class testname extends ActionSupport {
//通过从 Spring 的 ActionSupport 类而不是 Struts 的 Action 类进行扩展，创建了一个新的
Action
  public ActionForward execute(
    ActionMapping mapping,
    ActionForm form,
    HttpServletRequest request,
    HttpServletResponse response)
    throws IOException, ServletException {
    ...
    ApplicationContext ctx = getWebApplicationContext();
    //使用 getWebApplicationContext() 方法获得一个 ApplicationContext
    ctx.getBean("javabean");
    //查找一个 Spring bean
    ...
  }
    ...
  }
}
```

这时，struts-config.xml 配置结果格式如下：

```
<?xml version="1.0" encoding="ISO 8859 1" ?>
```

```
<!DOCTYPE struts config PUBLIC
        "-//Apache Software Foundation//DTD Struts Configuration 1.1//EN"
        "http://jakarta.apache.org/struts/dtds/struts-config 1 1.dtd">
<struts-config>
 <form-beans>
   <form-bean name="beanname" type="org.apache.struts.validator.
   DynaValidatorForm">
            <form-property name="beannameitem " type="java.lang.String"/>
   </form-bean>
  </form-beans>
 <global-forwards type="org.apache.struts.action.ActionForward">
    <forward  name=" "  path="/ "/>
    ...
 </global-forwards>
 <action-mappings>
   <action path="/ " forward="/WEB-INF/pages/*.htm"/>
   ...
   </action>
 </action-mappings>
 <message-resources parameter="ApplicationResources"/>
 <plug-in className="org.apache.struts.validator.ValidatorPlugIn">
   <set-property property="pathnames"
value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
 </plug-in>
 <plug-in className="org.springframework.web.struts.ContextLoaderPlugIn">
   <set-property property="contextConfigLocation" value="/WEB-INF/
   beans.xml"/>
 </plug-in>
</struts-config>
```

web.xml 配置格式：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app 2 2.dtd">
<web-app>
    <!-- Standard Action Servlet Configuration (with debugging) -->
    <servlet>
        <servlet-name>action</servlet-name>

<servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
        <init-param>
                <param name>application</param name>
                <param value>ApplicationResources</param value>
             </init param>
```

```
        <!-- struts configs -->
        <init-param>
            <param name>config</param name>
            <param-value>/WEB-INF/struts-config.xml</param-value>
        </init-param>
        <init-param>
            <param-name>debug</param-name>
            <param-value>0</param-value>
        </init-param>
        <init-param>
            <param-name>detail</param-name>
            <param-value>0</param-value>
        </init-param>
        <load-on-startup>2</load-on-startup>
    </servlet>
    <!-- Standard Action Servlet Mapping -->
    <servlet-mapping>
        <servlet-name>action</servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>
<!-- Welcome File List -->
<welcome-file-list>
<welcome-file>index.jsp</welcome-file>
<welcome-file>index.html</welcome-file>
<welcome-file>index.htm</welcome-file>
</welcome-file-list>
    <!-- Struts Tag Library Descriptors -->
    <!--
    <taglib>
        <taglib-uri>/tags/struts-bean</taglib-uri>
        <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
    </taglib>
    <taglib>
        <taglib-uri>/tags/struts-html</taglib-uri>
        <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
    </taglib>
    <taglib>
        <taglib-uri>/tags/struts-logic</taglib-uri>
        <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
    </taglib>
    <taglib>
        <taglib-uri>/tags/struts-nested</taglib-uri>
        <taglib-location>/WEB-INF/struts-nested.tld</taglib-location>
    </taglib>
    -->
</web-app>
```

7.3.2.2　使用 Spring 的 DelegatingRequestProcessor 覆盖 Struts 的 RequestProcessor

将 Spring 从 Struts 动作中分离是一个更巧妙的设计选择，其分离的一种方法就是使用 org.springframework.web.struts.DelegatingRequestProcessor 类来覆盖 Struts 的 RequestProcessor 处理程序。DelegatingRequestProcessor 方法的确比第一种方法好，但是仍然存在一些问题。如果使用一个不同的 RequestProcessor，则需要手动整合 Spring 的 DelegatingRequestProcessor。添加的代码会造成维护的麻烦并且将来会降低应用程序的灵活性。此外，还有过一些使用一系列命令来代替 Struts RequestProcessor 的传闻。这种改变将会对这种解决方法的生命周期造成负面的影响。如下程序代码通过 Spring 的 DelegatingRequestProcessor 进行整合的配置（struts-config.xml）方法：

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE struts-config PUBLIC
        "-//Apache Software Foundation//DTD Struts Configuration 1.1//EN"
        "http://jakarta.apache.org/struts/dtds/struts-config 1 1.dtd">
<struts-config>
 <form-beans>
    ...
  </form-beans>
 <global-forwards type="org.apache.struts.action.ActionForward">
     ...
 </global-forwards>
 <action-mappings>
    ...
 </action-mappings>
 <message-resources parameter="ApplicationResources"/>
 <controller processorClass="org.springframework.web.struts.
 DelegatingRequestProcessor"/>
/*利用 <controller> 标记来用 DelegatingRequestProcessor 覆盖默认的 Struts
RequestProcessor*/
 <plug-in className="org.apache.struts.validator.ValidatorPlugIn">
    <set-property property="pathnames"
      value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
 </plug-in>
 <plug-in className="org.springframework.web.struts.ContextLoaderPlugIn">
    <set-property property="csntextConfigLocation" value="/WEB-
    INF/beans.xml"/>
 </plug-in>
</struts-config>
```

接着进行 Spring 配置格式的注册该动作：

```
<?xml version "1.0" encoding "UTF 8"?>
<!DOCTYPE beans PUBLIC " //SPRING//DTD BEAN//EN"
  "http://www.springframework.org/dtd/spring beans.dtd">
```

```
<beans>
  <bean id "beanname" class "com.test.beannameImpl"/>
  <bean name-"/"
    class=" "> /*使用名称属性注册了一个 bean，以匹配 struts-config 动作映射名称*/
     <property name=" ">
        <ref bean=" "/>
     </property>
  </bean>
</beans>
```

最后通过“<bean name="/"”揭示了一个 JavaBean 属性，允许 Spring 在运行时填充属性。程序片段格式如下：

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.DynaActionForm;
public class testclassname extends Action {
  ...
  }
  public void testreslut(classnameinstance insname) {
    this.insname = insname;
```

//创建了一个 JavaBean 属性，DelegatingRequestProcessor 自动地配置这种属性。这种设计使 Struts 动作并不知道它正被 Spring 管理，并且使能够利用 Sping 的动作管理框架的所有优点。由于 Struts 动作注意不到 Spring 的存在，所以不需要重写 Struts 代码就可以使用其他控制反转容器来替换掉 Spring。其程序片断如下：

```
}
public ActionForward execute(
  ActionMapping mapping,
  ActionForm form,
  HttpServletRequest request,
  HttpServletResponse response)
  throws IOException, ServletException {
  ...
}
```

```
    ...
  }
}
```

这种方法的 web.xml 配置方法与第一种方法相同。

7.3.2.3　将 Struts Action 管理委托给 Spring 框架

第三种方法是将 Struts 动作管理委托给 Spring，其可以通过在 struts-config 动作映射中注册一个代理来实现。代理负责在 Spring 环境中查找 Struts 动作。由于动作在 Spring 的控制之下，所以它可以填充动作的 JavaBean 属性，并为应用诸如 Spring 的 AOP 拦截器之类的特性带来了可能。动作委托解决方法是这三种方法中最好的，因为 Struts 动作不了解 Spring，不对代码作任何改变就可用于非 Spring 应用程序中。RequestProcessor 的改变不会影响它，并且它可以利用 Spring AOP 特性的优点。如下程序代码片段是在 struts-config.xml 中配置 Struts 与 Spring 整合的委托方法：

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
...
<struts-config>
 <form-beans>
    ...
    </form-bean>

  </form-beans>
 <global-forwards type="org.apache.struts.action.ActionForward">
    ...
 </global-forwards>
 <action-mappings>
    <action  path="/welcome" forward="/WEB-INF/pages/*.htm"/>
    <action  path="/searchEntry" forward="/WEB-INF/pages/*.jsp"/>
    <action  path="/searchSubmit"
            type="org.springframework.web.struts.DelegatingActionProxy"
            /*注册 Spring 代理类的名称，而不是声明动作的类名。DelegatingActionProxy
            类使用动作映射名称查找 Spring 环境中的动作。这就是我们使用 ContextLoad-
            erPlugIn 声明的环境*/
            input="/*.do"
            validate="true"
            name=" ">
            <forward name="success" path="/WEB-INF/pages/detail.jsp"/>
            <forward name="failure" path="/WEB-INF/pages/search.jsp"/>
    </action>
 </action-mappings>
 ...
</struts config>
```

接着利用动作映射使用<bean>标记的名称属性简单地创建了一个 bean。这个动作的 JavaBean 属性像任何 Spring bean 一样被填充，程序片段格式如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
 "http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
  <bean id="" class=""/>
  <bean name="/ "
        class=" ">
    <property name=" ">
      <ref bean=" "/>
    </property>
  </bean>
</beans>
```

最后为在 Spring 配置文件中注册拦截器，这样可以将 Spring 的 AOP 拦截器应用于 Struts 动作。并通过将 Spring 拦截器应用于 Struts 动作，从而可以用最小的代价处理横切关注点，如下程序片段就是在 Spring 配置文件中注册拦截器（beans.xml）：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
  "http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
  <bean id="bookService" class="ca.nexcel.books.business.BookServiceImpl"/>
  <bean name="/searchSubmit"
        class="ca.nexcel.books.actions.SearchSubmit">
    <property name="bookService">
      <ref bean="bookService"/>
    </property>
  </bean>
  <!--  Interceptors -->
  <bean name="logger"
    class="ca.nexcel.books.interceptors.LoggingInterceptor"/>
    /*注册拦截器*/
  <!-- AutoProxies -->
  <bean name="loggingAutoProxy"
        class="org.springframework.aop.framework.autoproxy.
          BeanNameAutoProxyCreator">
   /*创建了一个 bean 名称自动代理，它描述如何应用拦截器。还有其他的方法定义拦截点，这种
   方法常见而简便*/
    <property name="beanNames">
          <value>/ </value>
/*将 Struts 动作注册为将被拦截的 bean。如果想要拦截其他的 Struts 动作，即只须要在
"beanNames" 下面创建附加的 <value> 标记*/
    </property>
```

```
    <property name "interceptorNames">
        <list>
          <value>logger</value>/*当拦截发生时，就执行 bean*/
        </list>
    </property>
  </bean>
</beans>
```

7.3.3　Struts 与 PP 整合

在本章的 7.2 节中，已经完成了 PP 的整合，现只需将 Struts 与 Portlet 整合就可以实现 DWR 完成 AJAX 与 Struts 通信。首先通过对 web.xml 的配置，完成对业务逻辑的调用，配置如下：

```
<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>/WEB-INF/classes/applicationContext-hibernate.xml</param-val
ue>
</context-param>
<servlet id="servlet 12345">
<servlet-name>SpringContextServlet</servlet-name>
<servlet-class>org.springframework.web.context.ContextLoaderServlet</serv
let-class>
<load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>SpringContextServlet</servlet-name>
<url-pattern>/*</url-pattern>
</servlet-mapping>
```

通过这样的配置，会首先初始化 Spring 框架自带的 ContextLoaderServlet，这个 Servlet 的作用就是读取由 contextConfigLocation 指定的 Spring 配置文件的位置，初始化 Spring 框架的 Context 对象，并将这个对象保存在 ServletContext 中，留待 Action 调用。

如下是对 portlet.xml 文件的配置格式，以实现 Struts 与 portlet 通信：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE portlet-app-def PUBLIC "-//TEST//DTD Portlet Application 1.1//EN"
"portlet 1.1.dtd">
<portlet >
    <portlet-app uid="sample.SamplePortlet.te" major-version="1" minor-
    version="0">
        <portlet-app-name>Sample application</portlet-app-name>
        <portlet id="sample.SamplePortlet"
            href "WEB INF/web.xml#test" major version "1" minor version "0">
            <portlet name>Sample portlet</portlet name>
```

```
                <cache>
                    <expires>0</expires>
                    <shared>no</shared>
                </cache>
                <allows>
                    <maximized />
                    <minimized />
                </allows>
                <supports>
                    <markup name="html">
                        <view />
                    </markup>
                </supports>
            </portlet>
        </portlet-app>
        <concrete-portlet-app uid="sample.SamplePortlet.tel">
            <portlet-app-name>Sample application</portlet-app-name>
            <concrete-portlet href="#sample.SamplePortlet">
                <portlet-name>Sample portlet</portlet-name>
                <default-locale>zh</default-locale>
                <language locale="zh">
                    <title>Sample portlet</title>
                    <title-short></title-short>
                    <description></description>
                    <keywords>WPS, Struts</keywords>
                </language>
                <config-param>
                    <param-name>FilterChain</param-name>
                    <param-value>StrutsTranscoding</param-value>
                </config-param>
            </concrete-portlet>
        </concrete-portlet-app>
    </portlet>
```

7.3.4 Spring 与 Hibernate 整合

Spring 框架提供了对 Hibernate、JDO 和 iBATIS SQL Maps 的集成支持。Spring 对 Hibernate 的支持是第一级的，整合了许多 IOC 的方便特性，解决了许多典型的 Hibernate 集成问题。并且框架对 Hibernate 的支持符合 Spring 通用的事务和数据访问对象（DAO）异常层次结构。Spring 为使用选择的 OR 映射层来创建数据访问应用程序提供了支持，因为所有业务逻辑都设计成一组可重用 JavaBean，所以不管选择什么技术，都能以库的格式访问大多数 Spring 的 OR 映射支持。ApplicationContext 或 BeanFactory 内部的 OR 映射的好处是简化了配置和部署。最终 Hibernate 使用 XML（*.hbm.xml）文件把 Java 类映射到表，把 JavaBean 属性映射

到数据库表。如下程序代码是实现 JDBC DataSource 和 HibernateSessionFactory 连接的格式（applicationContext-hibernate.xml）：

```
<!-- DataSource Property -->
<bean id="exampleDataSource"
    class="org.apache.commons.dbcp.BasicDataSource">
 <property name="driverClassName">
   <value> oracle.jdbc.driver.OracleDriver </value>
 </property>
 <property name="url">
   <value>jdbc: Oracle:springexample;create=true</value>
 </property>
</bean>
/*mySQL 连接方法
<property name="driverClassName">
            <value>com.mysql.jdbc.Driver</value>
        </property>
        <property name="url">
        <value>jdbc:mysql://localhost:3306/infos</value>
        </property>
        <property name="username">
            <value>root</value>
        </property>
        <property name="password">
            <value></value>
        </property>
*/
<!-- Database Property -->
<bean id="exampleHibernateProperties"
 class="org.springframework.beans.factory.config.PropertiesFactoryBean">
 <property name="properties">
  <props>
   <prop key="hibernate.hbm2ddl.auto">update</prop>
   <prop
    key="hibernate.dialect">net.sf.hibernate.dialect.DerbyDialect</prop>
   <prop
    key="hibernate.query.substitutions">true 'T', false 'F'</prop>
   <prop key="hibernate.show sql">false</prop>
   <prop key "hibernate.c3p0.minPoolSize">5</prop>
   <prop key "hibernate.c3p0.maxPoolSize">20</prop>
   <prop key="hibernate.c3p0.timeout">600</prop>
   <prop key="hibernate.c3p0.max statement">50</prop>
   <prop
     key-"hibernate.c3p0.testConnectionOnCheckout">false</prop>
  </props>
```

```
        </property>
    </bean>
    <!-- Hibernate SessionFactory -->
    <bean id="exampleSessionFactory"
      class="org.springframework.orm.hibernate.LocalSessionFactoryBean">
     <property name="dataSource">
        <ref local="exampleDataSource"/>
     </property>
     <property name="hibernateProperties">
        <ref bean="exampleHibernateProperties" />
     </property>
     <!--  OR mapping files. -->
     <property name="mappingResources">
       <list>
          <value>*.hbm.xml</value>
          ...
       </list>
     </property>
    </bean>
```

7.3.5 Spring 与 iBatis 整合

iBATIS 通过 SQL Map 将 Java 对象映射成 SQL 语句和将结果集再转化成 Java 对象，与其他 ORM 框架相比，既解决了 Java 对象与输入参数和结果集的映射，又能够让用户方便的手写使用 SQL 语句。同时，使用 iBATIS 的一个好处是 XML 配置让它得以成为一个很好的可用来将对象映射到现有关系数据库的 ORM 框架。有了 Mapper 类及映射文件，重点就变成了将对象映射到现有的数据结构，而不是使一个数据结构遵从这个对象的结构。在 iBATIS 3 的这个 XML 配置文件，其中指定了数据库的名称、要使用的驱动程序的类型以及其他的一些数据库属性，程序片段（ibatis-config.xml）如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
  PUBLIC "-//ibatis.apache.org//DTD Config 3.0//EN"
  "http://ibatis.apache.org/dtd/ibatis-3-config.dtd">
<configuration>
    <typeAliases>
        <typeAlias type="com.examples.ibatis.model.Automobile"
            alias="Automobile" />
    </typeAliases>
    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC" />
            <dataSource type="POOLED">
                <property name="driver"
```

```
                value "org.apache.derby.jdbc.EmbeddedDriver" />
            <property name "url" value "jdbc:derby:/tmp/MyDB" />
         </dataSource>
      </environment>
   </environments>
   <mappers>
      <mapper resource="automobile-mapper.xml" />
   </mappers>
</configuration>
```

iBATIS 框架的一个重要组成部分就是其 SqlMap 配置文件，SqlMap 配置文件的核心是 Statement 语句包括 CIUD。iBATIS 通过解析 SqlMap 配置文件得到所有的 Statement 执行语句，同时会形成 ParameterMap、ResultMap 两个对象用于处理参数和经过解析后交给数据库处理的 Sql 对象。同时，iBATIS 对管理事务既可以自己管理也可以由外部管理，iBATIS 自己管理是通过共享 SqlMapSession 对象实现的，多个 Statement 的执行时共享一个 SqlMapSession 实例，而且都是线程安全的。如果是外部程序管理就要自己控制 SqlMapSession 对象的生命周期，如图 7-1 所示。

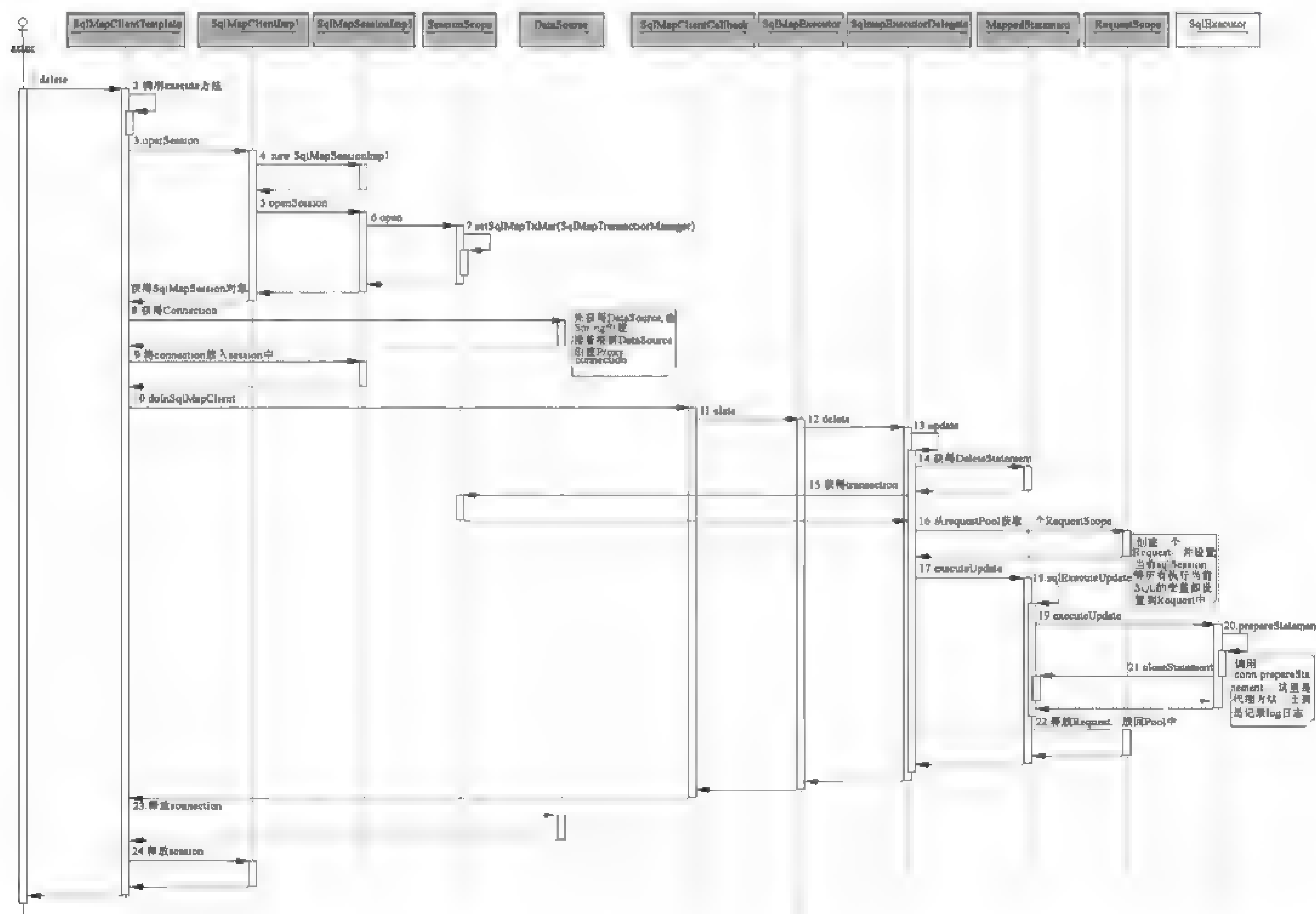

图 7-1　Spring 调用 iBATIS 执行一个 Statement 的时序图（IBM）

如下程序片段就是 Spring 与 iBATIS 集成的配置代码格式（applicationContext.xml）：

```
<bean id "dataSource"
      class "org.springframework.jdbc.datasource.DriverManagerDataSource">
```

```
        <property name "driverClassName">
            <value> oracle.jdbc.driver.OracleDriver </value>
        </property>
        <property name-"url">
            <value> jdbc: Oracle:springexample;create=true </value>
        </property>
        <property name="username">
            <value>sa</value>
        </property>
        <property name="password">
            <value></value>
        </property>
    </bean>
    <!-- ibatis sqlMapClient config -->
    <bean id="sqlMapClient"
        class="org.springframework.orm.ibatis.SqlMapClientFactoryBean">
        <property name="configLocation">
            <value>
        classpath:com\ibatis\jpetstore\persistence\sqlmapdao\sql\sql-map-
       config.xml
            </value>
        </property>
        <property name="dataSource">
            <ref bean="dataSource"/>
        </property>
    </bean>
    <!-- Transactions -->
    <bean id="TransactionManager"
        class="org.springframework.jdbc.datasource.DataSourceTransa-
        ctionManager">
        <property name="dataSource">
            <ref bean="dataSource"/>
        </property>
    </bean>
    <!-- persistence layer -->
    <bean id="AccountDao"
        class="com.ibatis.jpetstore.persistence.sqlmapdao.
        AccountSqlMapDao">
        <property name="sqlMapClient">
            <ref local="sqlMapClient"/>
        </property>
```

```
    </bean>
```

7.3.6 SSH 整合实现

SSH 的整合其实质是根据不同开源软件框架的特性，分别将开源软件框架放到不同的层次中去，即将 Struts 作为表示层、将 Spring 作为业务层及将 Hibernate/iBATIS 作为持久层。这样就可以根据的配置方法，就可以很容易完成对 SSH 及 SSi 的整合，其整合的载体通常是 Eclipse。

7.4 A2JT 融合

A2JT 是指将 Spring、CXF、OWL2OWL-S 实现整合来研发（语义）Web 服务应用程序，但由于 XML 与语义关系复杂，所涉及到数学基础困难，而且语义目前正处于发展阶段。因此，WSDL 与 WSDL2OWL-S 还很难实现自动转换，即通过配置文件来实现 WSDL 与 WSDL2OWL-S 自动转换是存在一定的困难。这时，就先将 CXF 与 Spring 整合，然后开发出 Web 服务，并发布这些 Web 服务，然后再在 OWL2OWL-S 图形界面中实现 WSDL 的语义化转换（见第 8 章），就可以完成 A2JT 融合。

同样，CXF 与 Spring 整合也是体现在基于 XML 的配置文件上，CXF 配置文件实际上是包含服务与客户端 Bean 定义的 Spring 配置文件。三个配置文件如下：

7.4.1 配置 web.xml 的格式

配置 web.xml 的目的是集成 Spring 和 CXF。其程序片断如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN" "http://java.sun.com/dtd/web-app 2 3.dtd">
<web-app>
    <display-name>Web Service Application with Spring and CXF</display-name>
    <description>Web Service Application with Spring and CXF</description>
    <!-- Spring Config Location -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/classes/beanRefServer.xml</param-value>
    </context-param>
    <!-- Spring ContextLoaderListener -->
    <listener>

<listener-class>org.springframework.web.context.ContextLoaderListener
</listener-class>
    </listener>
```

```
    <context param>
   <param name>contextConfigLocation</param name>
  /*
   <param-value>
     classpath:META-INF/cxf/*.xml
     </param-value>
 </context-param>
*/
    <!-- Apache CXFServlet -->
    <servlet>
        <servlet-name>CXFServlet</servlet-name>
        <display-name>CXF Servlet</display-name>
        <servlet-class>org.apache.cxf.transport.servlet.CXFServlet
        </servlet-class>
        <!-- <servlet-class> 是 CXF 代码的一部分而 <listener-class> 引用一个
        Spring Framework 类-->
        <load-on-startup>1</load-on-startup>
    </servlet>
    <!-- CXFServlet Mapping -->
    <servlet-mapping>
        <servlet-name>CXFServlet</servlet-name>
        <url-pattern>/*</url-pattern>
    </servlet-mapping>
</web-app>
```

7.4.2 配置 cxf-servlet.xml 的文件格式

一个独立文件 WEB-INF/cxf-servlet.xml 用于配置 CXF，使其将 servlet 接收的请求路由到服务实现代码并按需提供服务 WSDL。即创建 beans.xml，并使用 JAX-WS 前端将该服务类定义为 Spring Bean。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:jaxws="http://cxf.apache.org/jaxws"
     xmlns:soap="http://cxf.apache.org/bindings/soap"
     xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
        http://cxf.apache.org/jaxws
        http://cxf.apache.org/schemas/jaxws.xsd">
   <jaxws:endpoint
      id="Processor"
      implementor="com.test.ws.*.cxf.CXFtestImpl"
      wsdlLocation="WEB INF/wsdl/*.wsdl"
```

```
        address "/">
    </jaxws:endpoint>
    /*<beans> 元素仅是单个 bean 配置的一个包装器。<jaxws:endpoint>元素就是这样的一
    个 bean，CXF 通过特定类型的对象（一个 org.apache.cxf.jaxws.EndpointImpl 实例）
    与其相关联*/
</beans>
```

通常由于 Web 服务一般都具有服务与客户端应用程序的开发，这时就要分别对服务与客户的 bean.xml 进行配置。

（1）服务端的配置格式。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
xmlns=http://www.springframework.org/schema/beans xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" xmlns:jaxws="http://cxf.apache.
org/jaxws"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
        http://cxf.apache.org/jaxws
            http://cxf.apache.org/schemas/jaxws.xsd">
    <!-- Import Apache CXF Bean Definition -->
    <import resource="classpath:META-INF/cxf/cxf.xml"/>
    <import resource="classpath:META-INF/cxf/cxf-extension-soap.xml"/>
    <import resource="classpath:META-INF/cxf/cxf-servlet.xml"/>
    <!-- SurveyService -->
    <bean id="testService" class="ws.cxf.impl.testService">
        <property name="excludeName" value="testName"/>
        <property name="leastPonit" value="10"/>
    </bean>
    <!-- Expose testWebService -->
    <jaxws:server id="testWebService"
serviceClass="ws.cxf.ItestService" address="/testWebService">
        <jaxws:serviceBean>
            <ref bean="testService"/>
<!-- 要暴露的 bean 的引用 -->
        </jaxws:serviceBean>
    </jaxws:server>
</beans>
```

（2）客户端配置格式。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:jaxws="http://cxf.apache.org/jaxws"
```

```
        xsi:schemaLocation="
            http://www.springframework.org/schema/beans
                http://www.springframework.org/schema/beans/spring beans-2.0.xsd
            http://cxf.apache.org/jaxws
                http://cxf.apache.org/schemas/jaxws.xsd">
        <!-- Import Apache CXF Bean Definition -->
        <import resource="classpath:META-INF/cxf/cxf.xml"/>
        <import resource="classpath:META-INF/cxf/cxf-extension-soap.xml"/>
        <import resource="classpath:META-INF/cxf/cxf-servlet.xml"/>
        <!-- testWebService Client -->
        <jaxws:client id="testServiceClient" serviceClass="ws.cxf.ItestService"
    address="http://localhost:8080/CXF Spring test/testWebService"/>
</beans>
```

小　　结

本章进一步总结了开源软件框架整合的方法，这些内容主要包括通过 DWR 实现 AJAX 与 portlet（Jetspeed）通信，即完成 AJAX 与 portlet 的整合；总结了 Spring 与 Struts 的三种整合方法；实现了 Spring 与 Hibernate、AJAX、iBatis、CXF 整合等，从而为开发人员提供可供参与的配置方法。当然，除了本章整合方法以外，在本书的其他章节也列出其他的一些整合方法，如 Spring 与 Tuscany、portlet、Flex、Tuscany、Jetspeed、Liferay 等的整合。

第8章　SAJP-M轻量级开源中间件整合实现

本章将继续整合本书所涉及的开源软件框架，以形成一款满足多方业务逻辑需求的中间件，从而具备轻量级的软件研发要求，并用一个关于语义的例子进行应用。

8.1　SAJP-M概述

SAJP-M是由SSH、CXF、EXT、WSDL2OWL-S等开源软件框架整合而成，同时，根据各开源框架的特征还可以通过Spring继续整合Tuscany、Jetspeed、Flex等，其技术包括MVC、AOP、IoC、JSON、Web服务、SCA、portlet、OR等技术。这些内容本书都有详细叙述。从而通过基于XML的配置文件整合同一领域的开源软件框架，形成一个具有中间件特性的集成化系统。这可以有效的提供软件研发关键效率，增强软件的复用性，进一步满足不同层次的软件研发人员的要求。

SAJP-M 整合化中间件是以前面章节的软件构架为指导思想，并以叙述的开源软件相关技术为主要技术来实现的。这时，根据上面的分析，在某一领域中，开源软件的整合都可以以Spring Framework为核心来完成同一领域的开源软件集成。在本章中，是以Spring为核心，整合了Struts、CXF、WSDL2OWL等框架，如图8-1所示，以及集成AJAX、JOSN等新型框架和技术；并封装了hibernate数据访问，提供了常见的crud和分页等。只需继承一个DAO基类，就实现DAO层通信。同时，该框架提供的结构开发，能最大程度减少开发量，把所有代码开发关注在业务逻辑上，从而可以快速以该开发框架开发语义Web服务，为提供更有效、更灵活的语义Web服务组合去完成不同的业务逻辑需求。在应用程序开发时，该框架可以直接打成Jar包集成在其他项目中去应用。

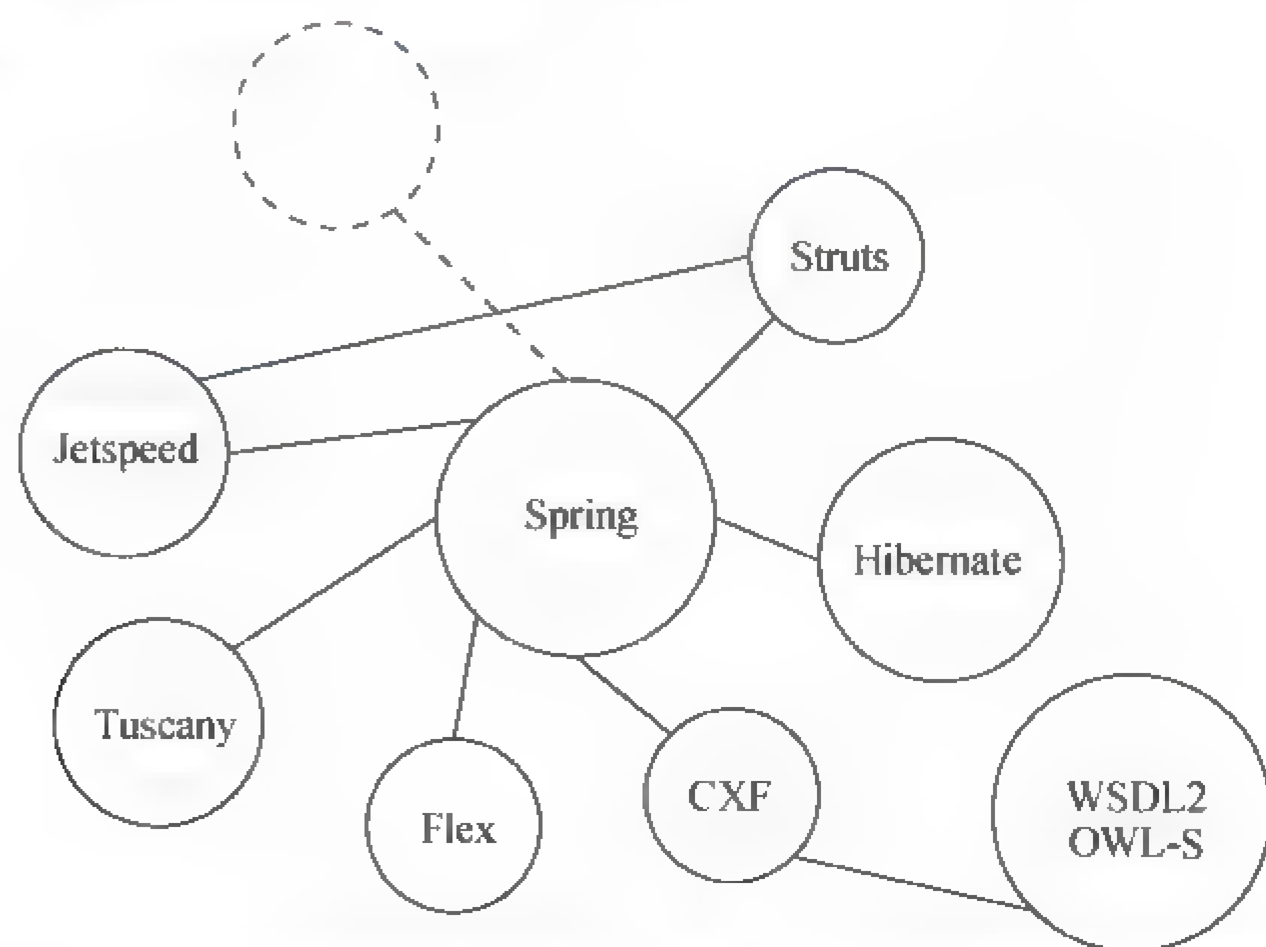

图8-1　以Spring为核心的开源软件整合模式（图中虚线表示其他可整合的开源软件）

8.2 SAJP-M 中间件结构

SAJP-M 就是将本书中相关开源软件通过基于 XML 的配置文件进行集成，当然在整合过程不仅是要进行配置，还需要通过需求变化开发不同的组件来简化编程流程。如本中间件不是简单的将 SSH 与之相关的开源软件通过 XML 配置整合就行了，而还开发 crud 和分页等，以及 DAO 基类来简化整个编程复杂性，以及通过 JSON 访问 JS 实现 EXT 呈现等。可以将一个开源软件可以认为一个构件，也可以将一个开源软件就是一个独立存在的软件实体；多个开源软件的集成就可以认为是构件组件过程。同时，面向服务的方法和技术应用，也可以将构件组装过程认为是服务组合过程。图 8-2 所示是本书相关开源软件整合结构。

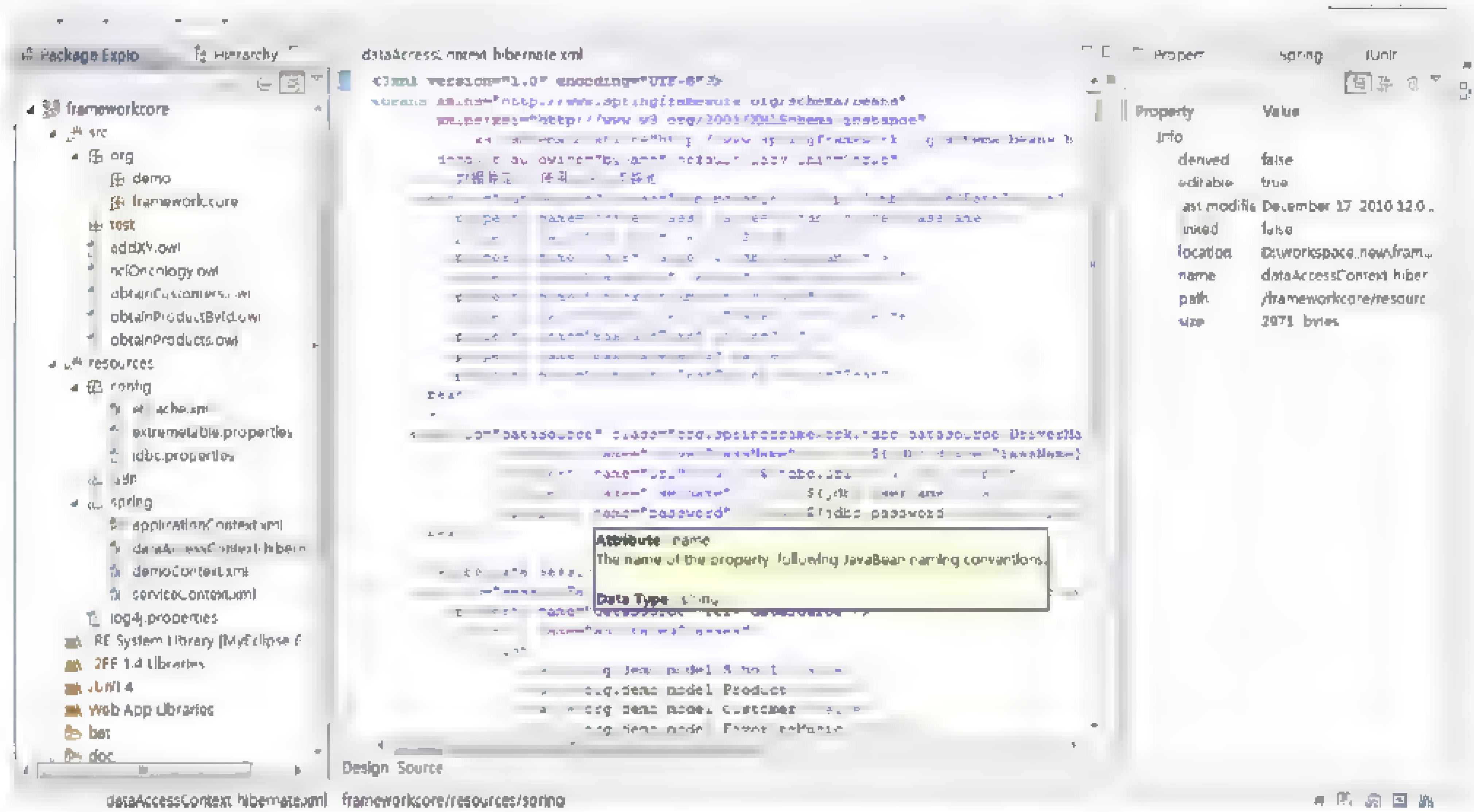

图 8-2　本书相关开源软件整合布局图

8.2.1 SAJP-M 主要的程序结构

研发 SAJP-M 的主要目的简化编程流程、提高软件研发效率、增强软件的可用性、提升软件可维护性等。图 8-3 所示是 SAJP-M 程序结构。

在图 8-3 中，对每一项简要简介如下：

（1）src 表示源码

（2）resources 包含资源，配置文件等

- config/jdbc.properties：数据库配置
- i18n：国际化文件配置
- spring : spring bean 文件的配置

（3）WebRoot : Web 项目的根

- commons：公共文件

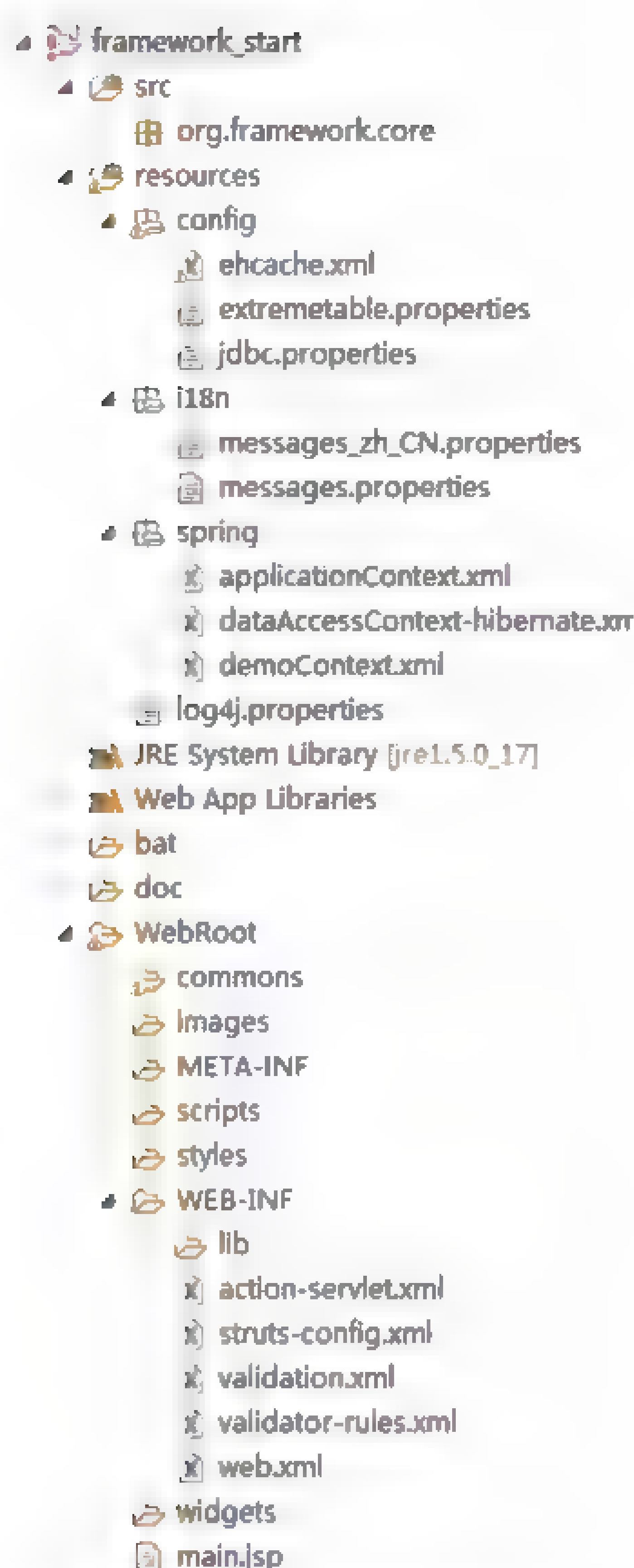

图 8-3　SAJP-M 程序结构

- images : 图片
- scripts : javascript 文件
- styles : 样式文件
- WEB-INF
    - lib : 项目依赖 jar
    - pages : jsp 文件
    - action-servlet.xml ： bean 配置文件
    - struts-config.xml : struts 配置文件
    - validation.xml : struts form 的验证配置
    - validation-rules.xml : 验证规则文件，为 validation.xml 使用
    - web.xml : 容器配置文件

下面针对以上程序结构的做如下简要分析：

8.2.1.1　数据处理

通常可以根据具体需求和对数据库要求来设计数据库的逻辑结构，并用 JPA Annoation 进行映射。或者用具体项目的流程直接设计 Entity 对象，然后生成数据库表结构。而且在设

计或创建过程中，Column 的命名尽量考虑可直接作为对象的属性名，例如 User Name 将被映射为属性 userName，这样有利于编程的统一性和可操作性。如下程序代码就是一个 Entity 对象的 JPA Annoatation 一对一、一对多等关系：

```
Class : org.demo.model.School
//@Entity 表示这是一个实体对象
//@Table(name = "T SCHOOL") 表示对应的数据库表名为 T SCHOOL
@Entity
@Table(name = "T SCHOOL")
public class School extends BaseEntity {
    private static final long serialVersionUID = -4271141611531328300L;
    private Long id;
    private String name;
    private String code;
    public void setId(Long id) {
        this.id = id;
    }
    //@Id 指明这是一个主键
    //@Column(name = "id", nullable = false) 对应表的 column 名为 id，并且不能为
      null
   //@GeneratedValue 可省略，指明那种主键映射方式，可以根据数据库选用和主键产生策略来
     确定
      @Id
    @Column(name = "id", nullable = false)
    @GeneratedValue
    public Long getId() {
        return id;
    }
    //column 名为 name
    @Column(name = "name")
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
   //column 名为 code
    @Column(name = "code")
    public String getCode() {
        return code;
    }
    public void setCode(String code) {
        this.code = code;
    }
```

同时，对于一个 Manager 对象管理多个紧密关联的 entity 对象，基本 CRUD 功能已经封装到了 DAO 基类的 BaseHibernateDao，无须再写，其他自定义方法就可以在子类中进行实现，示例代码如下：

```
Class : org.demo.mamager.SchoolManager
public class SchoolManager extends BaseHibernateDao<School> {
...
}
```

而对于 action 对象，继承于基类的 CRUD StrutsEntityAction 类，程序代码如下：

```
Class : org.demo.web.SchoolAction
public class SchoolAction extends StrutsEntityAction<School, SchoolManager>  {
    private SchoolManager schoolManager;
    public void setSchoolManager(SchoolManager schoolManager) {
        this.schoolManager = schoolManager;
    }
}
```

8.2.1.2　SSH 整合说明

SSH 的整合就是对 Hibernate、Spring、Struts 的配置，并根据具体的简化配置和编程的要求，实现持久层、业务层和表现层顺利访问。

1．Hibernate 的实现

Hibernate 实现的措施是采用 model 对象中的 JPA annotation 来代替了原来的*.hbm.xml；在 dataAccessContext-hibernate.xml 里定义了 sessionFactory，添加有 JPA 注释的 class 到 annotated Classes 参数里完成了实体的配置，从而使每增加一个实体的 CRUD 不需要修改一次公共配置文件。

2．Spring 的实现

Spring 的实现首先在 resources/spring 目录下，新建一个 xml 文件 demoContext.xml，并添加代码如下：

```
<bean id="schoolManager" class="org.demo.mamager.SchoolManager" />
```

这就表明 SchoolManager 这个类被 spring 管理了，即现在 SchoolManager 已经是个 bean 了。

接下来，在 WEB-INF 目录下的配置文件 action-servlet.xml 中添加代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "//SPRING//DTD BEAN 2.0//EN" "http:www.spring
-framework.org">
<beans default-autowire="byName" default-lazy-init="true">
  <bean name="/school" class="org.demo.web.SchoolAction" />
<beans>
```

把 SchoolAction 配置到 spring 中，由于<beans default-autowire="byName" default-lazy-

init="true">中的 default-autowire="byname"表示注入方式根据名字而对应。而在 SchoolAction 中，有一个 SchoolManager 的类属性成员，变量名为 schoolManager 的标量名。使得在 demoContext.xml 中已经定义的一个 bean，bean 的 id 就是 schoolManager。所以现在 bean id="schoolManager"的这个 bean 就会被注入到 SchoolAction 这个 bean 中，这时，只要 SchoolAction 有一个 javaBean 方式的 setter 方法就可以了，代码如下：

```
public void setSchoolManager(SchoolManager schoolManager) {
        this.schoolManager = schoolManager;
}//表示自动注入的条件
```

3．Struts 的实现

Struts 的实现首先在 WEB-INF 目录中找到 strust-config.xml 配置文件，进行配置如下：

```
<form-beans>
   <form-bean
            name="schoolForm" type="org.apache.struts.validator.
            LazyValidatorForm">
            <form-property name="name" type="java.lang.String" />
            <form-property name="code  type="java.lang.Integer" />
   <form-bean>
 </form-beans>
```

程序代码表示一个 form 表单对象，用于页面提交 form，绑定 from 表单里的字段到页面。并且其中 type="org.apache.struts.validator.LazyValidatorForm" 使用的是延迟 form，这个优点是不用专门再写一个 form 的 class 类。

接着，对配置 action 如下：

```
<action-mappings>
   <action path="/school" name="schoolForm" scope="request" parameter=
   "method"Validate="flas">
      <forward name="listPage" path="/WEB-INF/pages/schoolList.jsp" />
      <forward name="edit" path="/WEB-INF/pages/schoolForm.jsp" />
      <forward name="success" path="/school.do?method=deQuery" redirect=
      "true" />
   </action>
</action-mappings>
```

对上面的程序代码解释如下：

（1）path="/school" 配置文件 action-servlet.xml 中定义的<bean name="/school" class="org.demo.web.SchoolAction" /> 相对应，当从浏览器通过路径/school 访问时，就会调用这个指定的 bean。

（2）name="schoolForm" 表示使用的 form 对象。

（3）scope="request"表示作用范围是 request。

（4）parameter="method" 表示是继承的 action 的 DispatchAction，当访问/school 时，可以根据一个参数 method=<方法名>来实现所要执行的对应方法。

（5）validate="false"表示是否验证 form 表单，若是 true 表明验证，若是 false 表示不验证。

对于<forward name="listPage" path="/WEB-INF/pages/schoolList.jsp"/>，当 action 跳转到 listPage 时，就访问/WEB-INF/pages/schoolList.jsp 进行分页处理。其中 listPage 是当前框架内部定义的名字，当访问/school.do?method=doQuery 时，就会跳到 listPage 对应的页面<forward name="edit" path="/WEB-INF/pages/schoolForm.jsp"/>，其中 edit 是基类中定义的名字，当需要编辑一个页面就跳转位置<forward name="success" path="/school.do?method=doQuery" redirect="true"/>。Success 也是基类定义的名字，当保存或编辑成功后，就跳转到对应的位置。如下配置代码是一个跳转到 action 的例子，其中 redirect=true 表示页面重定向方式跳转：

```
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
   <set-property property="pathnames"
         value="/WEB-INF/validator-rules.xml, /WEB-INF/validation.xml" />
</plug-in>
```

其中，value="/WEB-INF/validator-rules.xml, /WEB-INF/validation.xml"表示指明指明验证文件。

最后，在 WEB-INF 下找到 validation.xml 进行配置如下：

```
<form name="schoolForm">
   <field property="name" depends="required">
      <arg key="school.name" />
   </field>
   <field property="code" depends="integer">
      <arg key="school.code" />
   </field>
</form>
```

配置文件中 form 对象中项 schoolForm 表示进行验证配置，name 字段 depends="required" 表示必须且不可缺少；depends="integer"表示整数，则写成 depends="required、integer"表示是必须的、不可缺少且为整数；<arg key="school.name"/> 为国际化消息的错误输出。

同时，在 resource/i18n 目录里，有 messages.properties 和 messages_zh_CN.properties 两个文件，第一个表示英语，第二个表示中文，里面会有对应的 key 和内容例如下面的代码：

```
school.name = School Name   //messages.properties
school.name = 学校名称       //messages_zh_CN.properties
```

这时，当 form 表单验证出错，就会输出以上错误消息。以上信息就是为实现 CRUD、分页等操作的配置方法。

4．数据库配置

数据库配置采用 JDBC 进行描述，首先在 resources/config 中找到 jdbc.properties 后配置如下：

```
//驱动名
jdbc.driverClassName com.mysql.jdbc.Driver
```

```
//jdbc url,demo 为数据库名，后面参数?createDatabaseIfNotExist true
表示当 demo 数据库不存在时，自动创建
jdbc.url=jdbc:mysql://localhost/demo?createDatabaseIfNotExist=true
//数据库用户名
jdbc.username=root
//数据库密码
jdbc.password=root
```

这时，对于安装 mysql 数据库由于 resources/spring/dataAccessContext-hibernate.xml 中，就可配置为<prop key="hibernate.hbm2ddl.auto">update</prop>，表明在系统启动时，会根据 entity 模型自动创建表结构，所以不需要生成表的 ddl。

5．事务管理配置

在对事务管理配置时，首先下 resources/spring 目录找到 applicaitonContext.xml 配置文件，配置如下：

```
<tx:annotation-driven>
<aop:aspectj-autoproxy>
<aop:config proxy-target-class="true">
   <aop:advisor pointcut="execution(*.demo.manager..*Manager.*(..)) "
                          advice-ref="txAdvice" />
   <aop:advisor pointcut="execution(*.demo.framework.core..*Dao.*(..)) "
                          advice-ref="txAdvice" />
   </aop:config>
```

在配置代码中，pointcut 指出对那些范围的包或 class 进行事务的操作，execution (*org.demo.manager..*Manager.*(..))表示对 org.demo.manager 下所有的类、类名后部分还有 Manager 的 class 的所有方法进行事务的控制。

8.2.1.3　A2JT 整合实现

在此小节，进一步描述 CXF、Spring、WSDL2OWL-S（此处不再描述 Tuscany 与 Spring 整合了，前面章节已经详细描述过；也不描述 Lucene、Kettle，其直接导入到环境平台中就可以整合开发）整合方法，并且在后面的应用中还将描述通过 JSON 去读取 OWL 和 OWL-S 数据的方法。其整合的程序结构如图 8-4 所示：

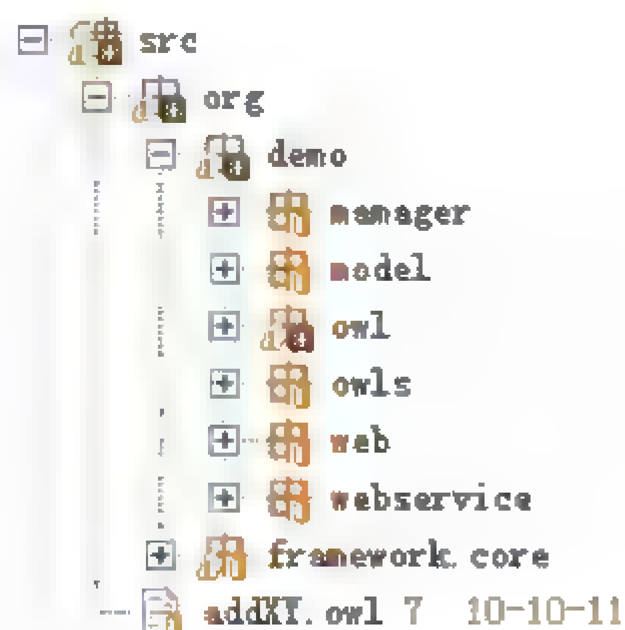

图 8-4　A2JT 整合后的结构结构

要实现 Web 服务的开发，

（1）得实现 CXF 与 Spring 的整合，在 web.xml 中进行 servlet 的配置，即定义 servlet：

```
<servlet>
  <servlet-name>CXFServlet</servlet-name>
  <servlet-class>org.apache.cxf.transport.servlet.CXFServlet
  </servlet-class>
  <load-on-startup>2</load-on-startup>
</servlet>
```

并且定义 servlet 对应的 url，在 ws 下的 url 都会被 CXFServlet 拦截，

```
<servlet-mappings>
    <servlet-name>CXFServlet</servlet-name>
    <url-pattern>/ws/*</url-pattern>
</servlet-mappings>
```

（2）在 resources/spring/applicationContext.xml 配置文件中添加代码如下：

```
<import resource="classpath:META-INF/cxf/cxf.xml" />
<import resource="classpath:META-INF/cxf/cxf-extension-soap.xml" />
<import resource="classpath:META-INF/cxf/cxf-servlet.xml" />
```

这表明导入 cxf 的资源文件。

接下来，配置 CXF 日志：

```
<cxf:bus>
    <cxf:features>
        <cxf:logging />
    </cxf:features>
</cxf:bus>
```

最后定义 CXF 的数据绑定格式：

```
<bean id="aegisBean"
        class="org.apache.cxf.aegis.databinding.AegisDatabinding"/>
<bean id="jaxws-and-aegis-service-factory"
class="org.apache.cxf.jaxws.support.JaxWsServiceFactoryBean" scope="prototype">
        <property name="dataBinding" ref="aegisBean" />
</bean>
```

（3）定义 Web 服务接口，其代码如下：

@WebService()表示定义 interface 为 webservice。

@SOAPBinding()定义 style 为 RPC，此方式 axis1.4 也可以通信。如果是 webservice2.0 或 wsdl1.2 版本以上，就可以不使用@SOAPBing()这个标记。

（4）实现这个 interface，代码如下：

```
package org.demo.webservice
```

```
public class AddFunctionImp implements IAddFunction{
...
}
```

（5）在 resource/spring/demoContext.xml 下配置 IAddFunction（具体配置方法参见前面 CXF 与 Spring 配置方法）。其目的是将 AddFunctionImpl 配置为 bean，id 为 addFunction。然后配置<jaxws>发布这个服务，并使 implement 指向要发布的 bean、addFunction。

（6）运行 WSDL2OWL-S，其运行配置（就是运行 bat 文件）的运行界面，如图 8-5 所示。

java -classpath ../WebRoot/WEB-INF/lib/owls-api-3.1-SNAPSHOT.jar;../WebRoot/WEB-INF/lib/iri-0.8.jar;../WebRoot/WEB-INF/lib/jena-2.6.3.jar;../WebRoot/WEB-INF/lib/arq-2.8.4.jar;../WebRoot/WEB-INF/lib/aterm-java-1.6.jar;../WebRoot/WEB-INF/lib/relaxngDatatype-20020414.jar;../WebRoot/WEB-INF/lib/xsdlib-20030225.jar;../WebRoot/WEB-INF/lib/commons-logging-1.1.jar;../WebRoot/WEB-INF/lib/pellet-2.1.1.jar;../WebRoot/WEB-INF/lib/jaxrpc-api-1.1.jar examples.WSDL2OWLS。

（7）将发布成的 WSDL 文件复制到图 8-5 中的“Enter URL”中，就可以实现语义转换了。

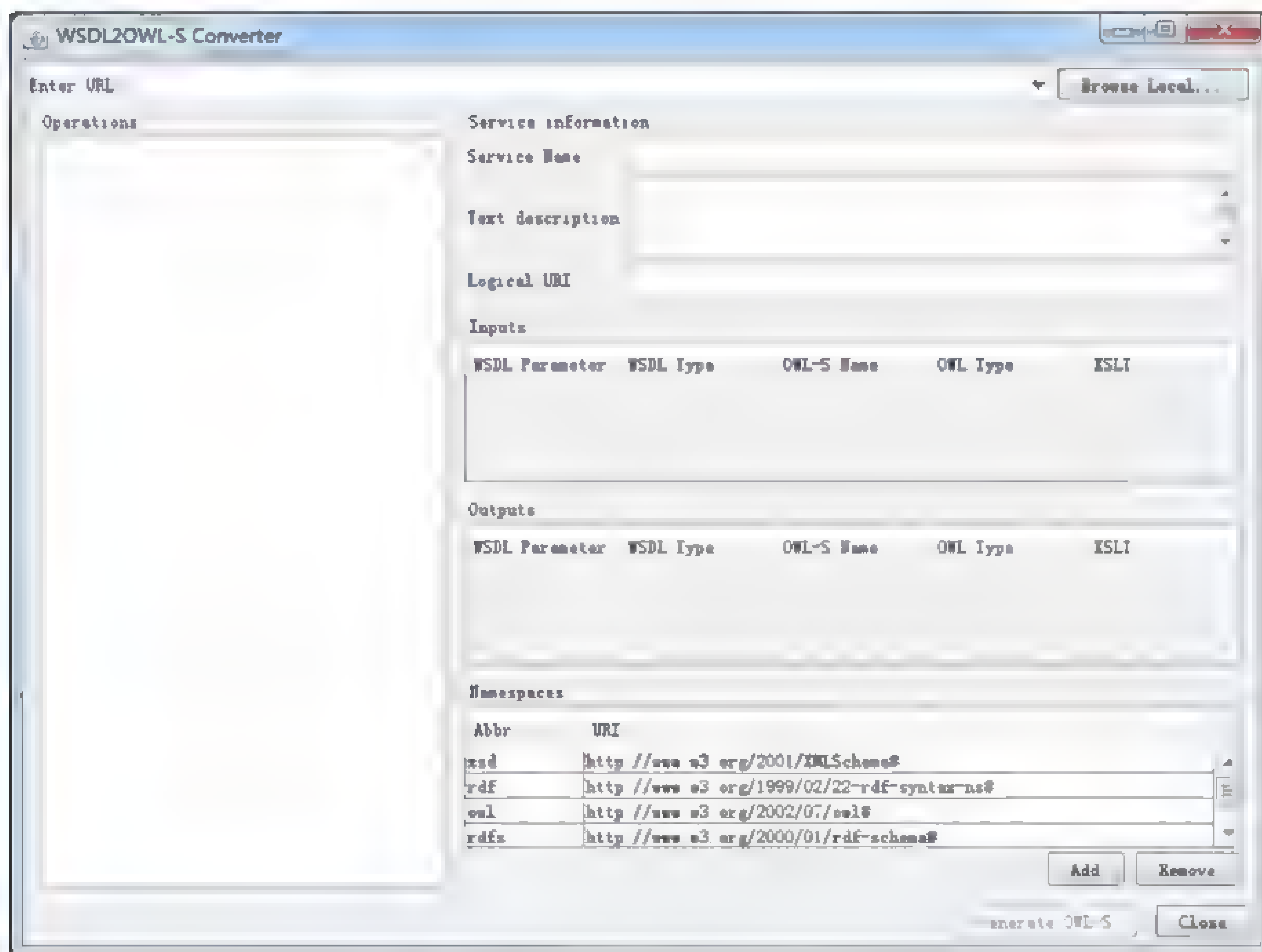

图 8-5 WSDL2OWL-S 运行界面

至此，基于 Spring 的语义 Web 服务开发就实现整合，也就可以方便实现了语义 Web 服务开发，即完成了 A2JT 整合实现。

8.2.2 SAJP-M 功能结构

整个 SAJP-M 中间件的功能结构源代码较多，但这些源代码多都是围绕 SAJP-M 中间整

合实现与应用举例（Demo）两个方面来完成的，而且整个代码分为服务端和客户端两类源代码；同时，从源代码类型角度来说，又可以分为配置文件源代码、核心源代码、JS 源代码、JSP 页面代码和 Demo 源代码。为了便于整体把握代码的情况，如表 8-1 所示。具体详细程序编码可以参见光盘代码。

表 8-1　SAJP-M 中间件主要配置文件

主要配置文件名	基本描述
applicationContext.xml	用于配置 Spring，且 Spring 与其他开源软件整合
dataAccessContext-hibernate.xml	配置 hibernate 及实现数据持久性
demoContext.xml	SAJP-M 的单独配置文件，实现 Spring 与 CXF 整合
serviceContext.xml	配置 Spring 中的 bean
action-servlet.xml	用来解决 Struts 与 Spring 开源软件整合配置文件
struts-config.xml	描述所有的 Struts 组件，用于配置 Struts 容器的配置文件
web.xml	Web 服务器页面配置文件
validation.xml	确认配置文件
validator-rules	确认规则配置文件

表 8-2　SAJP-M 中间件主要核心模块源代码

主要源文件名	基本描述
Constants.java	开源软件常量设置，有效控制开源件 JAR 包的读取
ConfigurableConstants.java	开源软件常量配置方法，实现资源定位，避免出错
AbstractUserType.java	用户抽象类属性
GenericEnumUserType.java	通用属性文件
UUIDTypeHandler.java	数据持久映射的通常标识
CriteriaSetup.java	用于判断框架的属性文件
StringConverter.java	通常字符串转换类，继承 Converter
ConfigurableConstants.java	常量配置文件
StrutsAction.java	Struts 的 Action 文件
StrutsEntityAction.java	Struts 的 EntityAction 文件
EntityInfo.java	判断 entity 是否 undeleteable 的标志
HibernateEntityExtendDao.java	Hibernate 的 Extend 数据访问对象
AbstractHibernateDao.java	Hibernate 的抽象数据访问对象
BaseHibernateDao.java	编写通用的 Hibernate，封装了 crud 和分页等功能
CustomCompactToolbar.java	继承了 CompactToolbar
CustomView.java	查看客户抽象类
AbstractEntityWithAuditColumns.java	数据持久化文件
BusinessException.java	业务逻辑属性文件
CriteriaPage.java	通常页面标准文件
HqlPage.java	创建页面文件
Page.java	分页实现文件
PageInfo.java	分页信息文件
BeanUtils.java	通用实体文件
ExtremeTablePage.java	分页排序文件
SpringUtil	通用性 Spring 类

表 8-3　基于 SAJP-M 中间件的 Demo 源代码

主要源代码名	基本描述
Customer.java	案例中的客户类
CustomerForm	案例中的客户 Form 类
FavoriteMusic.java	例子音乐选项类
Product.java	案例中基本信息类
School.java	案例中基本信息类
ClassesAndInstances.java	OWL 获取类
DataRanges.java	OWL 处理类
LoadingOntologies.java	加载 OWL 类
AddFunctionOWL.java	添加 OWL 功能类
CustomerOWL.java	客户 OWL 类
ProductOneOWL.java	基本信息 OWL 类
CustomerAction.java	客户信息的 Struts 的 Action 类，并通过 JSON 解析
CustomerDTO.java	客户基本信息属性
CustomerList.java	客户信息列表类
ProductAction.java	基本信息的 Struts 的 Action 类，并通过 JSON 解析
ProductDTO.java	基本信息的属性
ProductList.java	基本信息的列表类
SchoolAction.java	基本用户信息的 Struts 的 Action 类，并通过 JSON 解析
CustomerServiceImpl.java	客户信息实现类
ICustomerService.java	客户信息服务类
IProductService.java	基本信息服务类
ProductServiceImpl.java	基本信息服务实现类
obtainCustomers.owl	语义客户信息
obtainProductById.owl	语义化的个体基本信息
obtainProducts.owl	语义化的基本信息
customer_edit.js	用于 JSON 处理的客户编辑 JS 代码
customer_list.js	用于 JSON 处理的客户列表 JS 代码
customerEdit.jsp	客户信息编辑 JSP 页面
customerList.jsp	用户列表 JSP 页面
productEdit.jsp	基本信息编辑 JSP 页面
productList.jsp	基本信息列表 JSP 页面

表 8-4　是基于 SAJP-M 中间件应用的客户端源代码

主要源代码名	基本描述
CustomerOWLServiceImpl.java	语义 Web 服务实现类
ICustomerOWLService.java	定义服务接口
CustomerAction.java	获取语义 Web 服务数据并在客户端显示

8.3　应用方法

SAJP-M 中间件的应用包括了 SSH、A2JT 和界面开源框架 EXT 的基本应用，全面演示

了基本应用的步骤和流程，并且在第 9 章还将基于 SAJP-M 中间件开发一个科研绩效系统进一步较详细的说明。

8.3.1　SAJP-M 中间件主要配置文件

配置文件是开源软件框架整合重要操作之一，是将开源软件框架通过 J2EE 容器模式建立实现各开源软件通信。相关配置文件的说明已在本书进行了详细的解释，因此在此节就不再重复说明，只是列举出 SAJP-M 中间件所涉及到一些重要的配置文件。

8.3.1.1　applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:aop="http://www.springframework.org/schema/aop"
      xmlns:tx="http://www.springframework.org/schema/tx"
      xmlns:cxf="http://cxf.apache.org/core"
      xmlns:jaxws="http://cxf.apache.org/jaxws"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
      http://www.springframework.org/schema/aop
      http://www.springframework.org/schema/aop/spring-aop-2.5.xsd
      http://www.springframework.org/schema/tx
      http://www.springframework.org/schema/tx/spring-tx-2.5.xsd
      http://cxf.apache.org/core
      http://cxf.apache.org/schemas/core.xsd
      http://cxf.apache.org/jaxws
      http://cxf.apache.org/schemas/jaxws.xsd"
      default-autowire="byName" default-lazy-init="true">
    <!-- 属性文件读入 -->
    <bean id="propertyConfigurer" class="org.springframework.beans.factory.
    config. PropertyPlaceholderConfigurer">
        <property name="locations">
            <list>
                <value>classpath*:config/jdbc.properties</value>
            </list>
        </property>
    </bean>
        <!-- Load CXF modules from cxf.jar -->
    <import resource="classpath:META-INF/cxf/cxf.xml" />
    <import resource="classpath:META-INF/cxf/cxf-extension-soap.xml" />
    <import resource="classpath:META-INF/cxf/cxf-servlet.xml" />
    <!-- 打开 cxf 的日志  -->
    <cxf:bus>
        <cxf:features>
```

```
            <cxf:logging />
        </cxf:features>
    </cxf:bus>
    <!-- 定义数据绑定格式 -->
    <bean id="aegisBean"
        class="org.apache.cxf.aegis.databinding.AegisDatabinding"/>
    <bean id="jaxws-and-aegis-service-factory"
        class="org.apache.cxf.jaxws.support.JaxWsServiceFactoryBean"
        scope="prototype">
        <property name="dataBinding" ref="aegisBean" />
    </bean>
    <!--    <import resource="demoContext.xml"/>-->
    <!-- 支持 @Transactional 标记 -->
    <tx:annotation-driven/>
    <!-- 支持 @AspectJ 标记-->
    <aop:aspectj-autoproxy/>
    <!-- 以 AspectJ 方式 定义 AOP -->
    <aop:config proxy-target-class="true">
        <aop:advisor pointcut="execution(* org.demo.manager..*Manager.*
         (..))"advice-ref="txAdvice"/>
        <aop:advisor pointcut="execution(* org.framework.core..*Dao.*(..))"
        advice-ref="txAdvice"/>
    </aop:config>
    <!-- 基本事务定义，使用 transactionManager 作事务管理，默认 get*方法的事务为
    readonly，其余方法按默认设置。默认的设置请参考前一章内容 -->
    <tx:advice id="txAdvice" transaction-manager="transactionManager">
        <tx:attributes>
            <tx:method name="add*" propagation="REQUIRED"/>
            <tx:method name="del*" propagation="REQUIRED"/>
            <tx:method name="update*" propagation="REQUIRED"/>
            <tx:method name="save*" propagation="REQUIRED"/>
            <tx:method name="get*" propagation="SUPPORTS" read-only="true"/>
            <tx:method name="search*" propagation="SUPPORTS" read-only=
            "true"/>
            <tx:method name="find*" read-only="true"/>
            <tx:method name="*"/>
        </tx:attributes>
    </tx:advice>
</beans>
```

8.3.1.2 dataAccessContext-hibernate.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
        xsi:schemaLocation "http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring beans 2.5.xsd"
        default-autowire="byName" default-lazy init="true">
    <!-- 数据源定义，使用 C3P0 连接池
    <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"
    destroy-method="close">
        <property name="driverClass" value="${jdbc.driverClassName}"/>
        <property name="jdbcUrl" value="${jdbc.url}"/>
        <property name="user" value="${jdbc.username}"/>
        <property name="password" value="${jdbc.password}"/>
        <property name="checkoutTimeout" value="100"/>
        <property name="idleConnectionTestPeriod" value="60" />
        <property name="maxIdleTime" value="60" />
        <property name="maxStatements" value="100" />
        <property name="automaticTestTable" value="Test" />
    </bean>
     -->
    <bean id="dataSource" class="org.springframework.jdbc.datasource.
    DriverManager-DataSource">
            <property name="driverClassName">
<value>${jdbc.driverClassName}</value>
</property>
            <property name="url">
<value>${jdbc.url}</value>
</property>
            <property name="username">
<value>${jdbc.username}</value>
</property>
            <property name="password">
<value>${jdbc.password}</value>
</property>
    </bean>
    <!--Hibernate SessionFatory-->
    <bean id="sessionFactory" class="org.springframework.orm.hibernate3.
    anno -tation.AnnotationSessionFactoryBean">
        <property name="dataSource" ref="dataSource"/>
        <property name="annotatedClasses">
            <list>
                <value>org.demo.model.School</value>
                <value>org.demo.model.Product</value>
                <value>org.demo.model.Customer</value>
                <value>org.demo.model.FavoriteMusic</value>
            </list>
        </property>
        <property name "hibernateProperties">
```

```
            <props>
                <!--
<prop key="hibernate.dialect">
org.hibernate.dialect.SQLServerDialect
</prop>
                <prop key="hibernate.hbm2ddl.auto">update</prop>
                <prop key="hibernate.format sql">true</prop>
                <prop key="hibernate.show sql">true</prop>
                -->
                <prop key="hibernate.hbm2ddl.auto">update</prop>
                <prop key="hibernate.connection.provider class">
                    org.hibernate.connection.C3P0ConnectionProvider
                </prop>
                <prop key="connection.driver class">com.mysql.jdbc.Driver</prop>
                <prop key="hibernate.show sql">false</prop>
                <prop key="hibernate.dialect">
                    org.hibernate.dialect.MySQLDialect
                </prop>
                <prop key="hibernate.cache.provider class">
                    org.hibernate.cache.EhCacheProvider
                </prop>
                <prop key="hibernate.cache.use query cache">true</prop>
            </props>
        </property>
    </bean>
    <!--Hibernate TransactionManager-->
   <bean id="transactionManager" class="org.springframework.orm.hibernate3.
    HibernateTransactionManager">
        <property name="sessionFactory" ref="sessionFactory"/>
    </bean>
</beans>
```

8.3.1.3 demoContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:cxf="http://cxf.apache.org/core"
      xmlns:jaxws="http://cxf.apache.org/jaxws"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
         http://cxf.apache.org/core http://cxf.apache.org/schemas/core.xsd
         http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd"
      default-autowire="byName" default-lazy-init="true">
    <!-- cxf webservice pulbish  -->
    <jaxws:endpoint id="AddFunctionWS" implementor="#addFunction"
```

```
            address="/addFunction">
            <jaxws:serviceFactory>
                <ref bean="jaxws-and-aegis-service-factory" />
            </jaxws:serviceFactory>
        </jaxws:endpoint>
        <jaxws:endpoint id="ProductService" implementor="#productService"
            address="/productService">
            <jaxws:serviceFactory>
                <ref bean="jaxws-and-aegis-service-factory" />
            </jaxws:serviceFactory>
        </jaxws:endpoint>
        <jaxws:endpoint id="CustomerService" implementor="#customerService"
            address="/customerService">
            <jaxws:serviceFactory>
                <ref bean="jaxws-and-aegis-service-factory" />
            </jaxws:serviceFactory>
        </jaxws:endpoint>
        <!-- The service bean -->
        <bean id="addFunction" class="org.demo.webservice.AddFunctionImpl" />
        <bean id="productService" class="org.demo.webservice.ProductServi-
        ceImpl" />
        <bean id="customerService" class="org.demo.webservice.CustomerServ-
        iceImpl" />
        <!-- Manager bean -->
        <bean id="schoolManager" class="org.demo.manager.SchoolManager" />
        <bean id="productManager" class="org.demo.manager.ProductManager" />
        <bean id="customerManager" class="org.demo.manager.CustomerManager" />

    </beans>
```

8.3.1.4　serviceContext.xml

```
    <?xml version="1.0" encoding="UTF-8"?>
    <beans xmlns="http://www.springframework.org/schema/beans"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-2.5.xsd"
          default-autowire="byName" default-lazy-init="true">
    </beans>
```

8.3.1.5　action-servlet.xml

```
    <beans default-autowire="byName" default-lazy-init="true" >
        <bean name="/school" class="org.demo.web.SchoolAction"  />
        <bean name="/product" class="org.demo.web.ProductAction" />
        <bean name="/customer" class="org.demo.web.CustomerAction" />
```

```
</beans>
```

8.3.1.6 struts-config.xml

```
<struts-config>
    <form-beans>
        <form-bean name="schoolForm" type="org.apache.struts.validator.
        LazyValidatorForm">
            <form-property name="name" type="java.lang.String"/>
             <form-property name="code" type="java.lang.Integer"/>
        </form-bean>
        <form-bean name="productForm" type="org.apache.struts.validator.
        LazyValidatorForm">
            <form-property name="productName" type="java.lang.String"/>
             <form-property name="vender" type="java.lang.String"/>
             <form-property name="produceDate" type="java.util.Date"/>
        </form-bean>
        <form-bean name="customerForm" type="org.apache.struts.validator.
        LazyValidatorForm">
            <form-property name="name" type="java.lang.String"/>
             <form-property name="email" type="java.lang.String"/>
             <form-property name="birthday" type="java.util.Date"/>
             <form-property name="musics" type="java.lang.String[]"/>
             <form-property name="color" type="java.lang.String"/>
             <form-property name="fruit" type="java.lang.String"/>
             <form-property name="description" type="java.lang.String"/>
             <form-property name="duration" type="java.lang.Long"/>
        </form-bean>
    </form-beans>
    <action-mappings>
      <action path="/school" name="schoolForm" scope="request"
            parameter="method" validate="false">
            <forward name="listPage" path="/WEB-INF/pages/schoolList.jsp"/>
            <forward name="edit" path="/WEB-INF/pages/schoolForm.jsp"/>
            <forward name="success" path="/school.do?method=doQuery"
            redirect="true"/>
        </action>
        <action path="/product" name="productForm" scope="request"
            parameter="method" validate="false">
            <forward name="list" path="product.do?method=direct"
            redirect="true"/>
            <forward name-"direct"
            path-"/WEB-INF/pages/product/productList.jsp"/>
            <forward name-"edit"
            path "/WEB INF/pages/product/productEdit.jsp"/>
            <forward name "success" path-"product.do?method-direct"
```

```
            redirect="true"/>
        </action>
        <action path="/customer" name="customerForm" scope="request"
        parameter="method" validate="false">
            <forward name="list"
            path="customer.do?method=direct" redirect="true"/>
            <forward name="direct"
        path="/WEB-INF/pages/customer/customerList.jsp"/>
            <forward name="edit"
        path="/WEB-INF/pages/customer/customerEdit.jsp"/>
            <forward name="success"
        path="customer.do?method=direct" redirect="true"/>
        </action>
    </action-mappings>
    <controller>
        <set-property property="processorClass"
        value="org.springframework.web.struts.DelegatingRequestProcessor"/>
        <set-property property="maxFileSize" value="2M"/>
        <set-property property="inputForward" value="true"/>
    </controller>
    <message-resources parameter="i18n/messages"/>
    <plug-in className="org.springframework.web.struts.
    ContextLoaderPlugIn"/>
    <plug-in className="org.apache.struts.validator.ValidatorPlugIn">
        <set-property property="pathnames"
      value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
    </plug-in>
</struts-config>
```

8.3.1.7　web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.4"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.
         sun.com/xml/ns/j2ee/web-app_2_4.xsd" id="WebApp_1248800607637">
    <!-- Spring ApplicationContext 配置文件的路径，可使用通配符，多个路径用，号分隔
      此参数用于后面的 Spring-Context loader -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath*:spring/*.xml</param-value>
    </context-param>
    <!-- 默认 i18n 资源文件 -->
    <context-param>
        <param-name>javax.servlet.jsp.jstl.fmt.localizationContext</param-name>
        <param-value>i18n/messages</param-value>
```

```
    </context param>
    <context-param>
        <param name>extremecomponentsPreferencesLocation</param name>
        <param-value>/config/extremetable.properties</param-value>
    </context-param>
    <context param>
        <param-name>extremecomponentsMessagesLocation</param-name>
        <param-value>i18n/messages</param-value>
    </context-param>
        <!—注名 Character Encoding filter -->
        <filter>
        <filter-name>encodingFilter</filter-name>
        <filter-class>
          org.springframework.web.filter.CharacterEncodingFilter
          </filter-class>
        <init-param>
            <param-name>encoding</param-name>
            <param-value>UTF-8</param-value>
        </init-param>
    </filter>
    <filter>
        <filter-name>hibernateFilter</filter-name>
        <filter-class>
        org.springframework.orm.hibernate3.support.OpenSessionInViewFilter
        </filter-class>
    </filter>
    <filter-mapping>
        <filter-name>encodingFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
    <filter-mapping>
        <filter-name>hibernateFilter</filter-name>
        <url-pattern>*.do</url-pattern>
    </filter-mapping>
    <!--Spring ApplicationContext 载入 -->
    <listener>
        <listener-class>
        org.springframework.web.context.ContextLoaderListener
        </listener-class>
    </listener>
    <!-- Spring 刷新 Introspector 防止内存泄露 -->
    <listener>
        <listener-class>
        org.springframework.web.util.IntrospectorCleanupListener
        </listener-class>
```

```
    </listener>
    <!-- Struts Action Mapping -->
    <servlet>
        <servlet-name>action</servlet-name>
        <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
        <init-param>
            <param-name>config</param-name>
            <param-value>/WEB-INF/struts-config.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
        <servlet>
        <servlet-name>CXFServlet</servlet-name>
        <servlet-class>org.apache.cxf.transport.servlet.CXFServlet
        </servlet-class>
        <load-on-startup>2</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>action</servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>
        <servlet-mapping>
        <servlet-name>CXFServlet</servlet-name>
        <url-pattern>/ws/*</url-pattern>
    </servlet-mapping>
        <!-- session 超时定义，单位为分钟 -->
    <session-config>
        <session-timeout>30</session-timeout>
    </session-config>
    <!-- 默认首页定义 -->
    <welcome-file-list>
        <welcome-file>main.jsp</welcome-file>
    </welcome-file-list>
        <jsp-config>
        <jsp-property-group>
            <display-name>gggg</display-name>
            <url-pattern>*.jsp</url-pattern>
            <page-encoding>UTF-8</page-encoding>
            <include-prelude>/commons/pre include.jsp</include-prelude>
        </jsp-property-group>
    </jsp-config>
</web-app>
```

8.3.1.8　validation.xml

```
<?xml version "1.0" encoding "UTF 8"?>
```

```
<!DOCTYPE form validation PUBLIC
        "-//Apache Software Foundation//DTD Commons Validator Rules
        Configuration 1.3.0//EN"
        "http://jakarta.apache.org/commons/dtds/validator 1 3 0.dtd">
<form-validation>
    <formset>
        <form name="schoolForm">
            <field property="name" depends="required">
                <arg key="school.name"/>
            </field>
           <field property="code" depends="integer">
                <arg key="school.code"/>
            </field>
        </form>
        <form name="productForm">
            <field property="productName" depends="required">
                <arg key="product.productName"/>
            </field>
           <field property="vender" depends="required">
                <arg key="product.vender"/>
            </field>
            <field property="produceDate" depends="required">
                <arg key="product.produceDate"/>
            </field>
        </form>
    </formset>
</form-validation>
```

8.3.2 应用举例

基于 SAJP-M 中间件的基本应用是以客户偏好（以 demo 命名）为例进行描述。首先得配置 web.xml 和 applicationContext.xml，以实现程序的初始化（具体配置文件前一节配置文件内容）。其在 web.xml 中的 classpath 路径中所有位于 spring 文件夹下的以.xml 后缀结尾的文件都作为 context 的配置文件加载进来。而在 applicationContext.xml 中，对 config/jdbc.properties 配置，以实现数据库连接及调协用户名和密码，并加上了参数 createDatabaseIf-NotExist true，其目的是，当系统启动时，会检查数据库是否有 demo 数据库存在，如果不存在，会自动创建 demo 数据库。

8.3.2.1 实例业务逻辑

现在在 demo 下创建一个 Customer 实体类，它继承 org.framework.core.entity.BaseEntity，程序代码如下：

```
package org.demo.model;
```

```
import javax.persistence.Entity;
import javax.persistence.Table;
import org.framework.core.entity.BaseEntity;
@Entity
@Table(name = "t customer")
public class Customer extends BaseEntity {
   public Long getId()
    {
        return null
    }
}
```

在上面的程序片段中，打上 jpa 标记供 hibernate 控制。而@Entity 为一个实体类，@Table(name="t_customer") 表现与数据库中表 t_customer 关联。而整个 Customer 实体类如下程序代码清单 8-1，完成后运行结果如图 8-6 所示。

图 8-6　Customer 运行界面

从图中可以，包括了客户的基本信息和基本偏好，即 name、email、music、color、fruit、description。

代码清单 8-1　Customer.java：

```
package org.demo.model;
import java.util.Date;
import java.util.HashSet;
import java.util.Set;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
```

```
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.Table;
import javax.persistence.Transient;
import org.demo.web.MusicCheckBox;
import org.framework.core.entity.BaseEntity;
import org.springframework.util.StringUtils;
@Entity
@Table(name = "t customer")
public class Customer extends BaseEntity {
    private static final long serialVersionUID = -7423470663388802022L;
    private Long id;
    private String name;
    private String email;
    private Date birthday;
    private Set<FavoriteMusic> music;
    private String[] musics;
    private String color;
    private String fruit;
    private String description;
    private Double duration;
    public Customer() {
    }
    public Customer(String name, String email, Date birthday, Set
    <FavoriteMusic> music,
            String color, String fruit, String description, Double duration) {
    //客户基本信息类
        super();
        this.name = name;
        this.email = email;
        this.birthday = birthday;
        this.music = music;
        this.color = color;
        this.fruit = fruit;
        this.description = description;
        this.duration = duration;
    }
    @Id
    @Column(name = "id", nullable = false)
    @GeneratedValue
    public Long getId() {
        return id;
    }
    //表示主键是 id，对应的是表中的 id，@GeneratedValue 表示生成策略
    @Column(name - "name")
```

```
    public String getName() {
        return name;
    }
    //表示数据 name 映射表中 name 字段，下面共一格式的都是表示这个意思
    @Column(name = "email")
    public String getEmail() {
        return email;
    }
    @Column(name = "birthday")
    public Date getBirthday() {
        return birthday;
    }
    @OneToMany(mappedBy="customer",cascade=CascadeType.ALL, fetch=
    FetchType.EAGER)
    @org.hibernate.annotations.Cascade(org.hibernate.annotations.
    CascadeType.DELETE_ORPHAN)
//@OneToMany(mappedBy="customer") 表示多对一的 FavriteMusic 的属性 customer 关联，
  Cascade=CascadeType.All 表示级联所有，fetch=FethType.EAGER 表示预加载。
  @org.hibernate.annotations.Cascade(org.hibernate.annotations.CascadeType.
  DELETE_ORPHAN) 是 hibernate 特有的功能，但删除此对象时，对方引用的对象也被删除，这
  个特性 jpa 还没有加入
    public Set<FavoriteMusic> getMusic() {
        return music;
    }
 //在这里使用了 Struts 1 作为 mvc 层。Struts 1 提供的表单 form 绑定，对于 Set 集合是不
  支持的，只支持数组 List。所以，这里中间取巧用 String[] musics 与表单绑定，然后在转
  化为 Set<FavoriteMusic> music。当然 hibernate 也支持数组和 List 的映射，不过这里
  不需要数组的特性，不需要 index 的特性，也是因为 index 的特性需要额外的维护
    @Column(name = "color")
    public String getColor() {
        return color;
    }
    @Column(name = "fruit")
    public String getFruit() {
        return fruit;
    }
    @Column(name = "description")
    public String getDescription() {
        return description;
    }
    @Column(name = "duration")
    public Double getDuration() {
        return duration;
    }
    @Transient
```

```
    public String[] getMusics() {
        return musics;
    }
    public void setMusics(String[] musics) {
        this.musics = musics;
        Set<FavoriteMusic> favoriteMusics = getMusic();
        if (favoriteMusics==null) {
            favoriteMusics = new HashSet<FavoriteMusic>();
            setMusic(favoriteMusics);
        } else {
            favoriteMusics.clear();
        }
        if (musics!=null && musics.length!=0) {
            for (String music : musics) {
                if (StringUtils.hasText(music)) {
                    FavoriteMusic fm =
                    new FavoriteMusic(Music.valueOf(music),this);
                    favoriteMusics.add(fm);
                }
            }
        }
    }
   //在 String[] musics 被赋值时，就自动转化为 Set<FavoriteMusic> music
    private boolean isExistMusic(Set<FavoriteMusic> favoriteMusics, Music
    music) {
        if (null==favoriteMusics || favoriteMusics.isEmpty()) {
            return false;
        }
        for (FavoriteMusic fmusic : favoriteMusics) {
            if (fmusic.getMusic().equals(music)) {
                return true;
            }
        }
        return false;
    }
    @Transient
    public Set<MusicCheckBox> getMusicCheckBoxs(){
        Set<FavoriteMusic> favoriteMusics = getMusic();
        Set<MusicCheckBox> musicCheckBoxs = new HashSet<MusicCheckBox>();
        for (Music music : Music.values()) {
            boolean checked = false;
            if (isExistMusic(favoriteMusics,music)) {
                checked = true;
            }
            musicCheckBoxs.add(new MusicCheckBox(music, checked ));
```

```
        }
        return musicCheckBoxs;
    }
    //@Transient 表明 get 方法不会被 jpa 当做属性绑定。该方法返回一个 CheckBox 的集合，
      用于在页面方便的显示出那些 music 是被选中
    public void setId(Long id) {
        this.id = id;
    }
    public void setName(String name) {
        this.name = name;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public void setBirthday(Date birthday) {
        this.birthday = birthday;
    }
    public void setMusic(Set<FavoriteMusic> music) {
        this.music = music;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public void setFruit(String fruit) {
        this.fruit = fruit;
    }
    public void setDescription(String description) {
        this.description = description;
    }
    public void setDuration(Double duration) {
        this.duration = duration;
    }
}
```

与 Customer.java 的 MusicCheckBox.java 类、FavoriteMusic.java 类及枚举类请详见光盘代码。下面继续描述 CustomerManager 类，代码如下所示，并将其配置到 resource/spring demoContent.xml 中。

```
package org.demo.mamager;
import org.demo.model.Customer;
import org.framework.core.dao.BaseHibernateDao;
public class CustomerManager extends BaseHibernateDao<Customer>{

}
```

并介绍 BaseHibernateDao 类，为了简化程序编写，它封转了 crud、分页等方法，详见代

码清单 8-2，也可以详见光盘 BaseHibernateDao 代码。

代码清单 8-2 BaseHibernateDao.java：

```
package org.framework.core.dao;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import javax.servlet.http.HttpServletRequest;
import org.apache.commons.lang.StringUtils;
import org.extremecomponents.table.core.TableConstants;
import org.extremecomponents.table.limit.Limit;
import org.framework.core.Constants;
import org.framework.core.commons.EntityService;
import org.framework.core.commons.support.CriteriaSetup;
import org.framework.core.commons.support.FunctionName;
import org.framework.core.dao.extend.UndeleteableEntityOperation;
import org.framework.core.page.CriteriaPage;
import org.framework.core.page.HqlPage;
import org.framework.core.page.Page;
import org.framework.core.page.PageInfo;
import org.framework.core.util.BeanUtils;
import org.framework.core.util.ExtremeTablePage;
import org.framework.core.util.GenericsUtils;
import org.hibernate.Criteria;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.criterion.CriteriaSpecification;
import org.hibernate.criterion.Criterion;
import org.hibernate.criterion.Order;
import org.hibernate.criterion.Projection;
import org.hibernate.criterion.Projections;
import org.hibernate.criterion.Restrictions;
import org.hibernate.impl.CriteriaImpl;
import org.springframework.dao.DataAccessException;
import org.springframework.util.Assert;
import org.springframework.util.CollectionUtils;
abstract public class BaseHibernateDao<T> extends AbstractHibernateDao<T>
        implements EntityService<T>, UndeleteableEntityOperation<T> {
```

```
    public static int COUNT MODE = 1;
    public static int SCROLL MODE = 2;
    public static int LIST MODE = 3;
    public Integer FindTotalEntitys(T object, Map<String, String> filter) {
        Criteria criteria = getEntityCriteria();
        if (filter != null && !filter.isEmpty()) {
            Set<String> filterNameSet = filter.keySet();
            for (String name : filterNameSet) {
                criteria.add(Restrictions.like(name, "%" + filter.get(name)
                        + "%"));
            }
        }
        return (Integer) criteria.setProjection(Projections.rowCount())
                .uniqueResult();
    }
    @SuppressWarnings("unchecked")
    public List<T> FindEntitysForPage(int startRow, int countPerPage,
    T object,
            Map<String, String> order, Map<String, String> filter) {
        Criteria criteria = getEntityCriteria();
        if (order != null && !order.isEmpty()) {
            Set<String> orderNameSet = order.keySet();
            for (String name : orderNameSet) {
                if ("asc".equalsIgnoreCase(order.get(name))) {
                    criteria.addOrder(Order.asc(name));
                } else {
                    criteria.addOrder(Order.desc(name));
                }
            }
        }
        if (filter != null && !filter.isEmpty()) {
            Set<String> filterNameSet = filter.keySet();
            for (String name : filterNameSet) {
                criteria.add(Restrictions.like(name, "%" + filter.get(name)
                        + "%"));
            }
        }
        return criteria.setFirstResult(startRow).setMaxResults(
                countPerPage - startRow).list();
    }
    @SuppressWarnings("unchecked")
    public Page findBy(Map filterMap, int pageNo, int pageSize) {
        return findBy(filterMap, null, pageNo, pageSize);
    }
    @SuppressWarnings("unchecked")
```

```
    public Page findBy(Map filterMap, Map orderMap, int pageNo, int pageSize,
            CriteriaSetup criteriaSetup) {
        Criteria criteria = getSession().createCriteria(getEntityClass());
        if (!CollectionUtils.isEmpty(filterMap)) {
            try {
                criteriaSetup.setup(criteria, filterMap);
            } catch (Exception e) {
            }
        }
        if (!CollectionUtils.isEmpty(orderMap))
            sortCriteria(criteria, orderMap, null);
        criteria.setResultTransformer(CriteriaSpecification.ROOT ENTITY);
        return pagedQuery(criteria, pageNo, pageSize);
    }
    @SuppressWarnings("unchecked")
    public Page findBy(Map filterMap, Map orderMap, int pageNo, int pageSize,
            Criteria criteria) {
        CriteriaSetup criteriaSetup = getDefaultCriteriaSetup();
        if (!CollectionUtils.isEmpty(filterMap)) {
            try {
                criteriaSetup.setup(criteria, filterMap);
            } catch (Exception e) {
            }
        }
        if (!CollectionUtils.isEmpty(orderMap))
            sortCriteria(criteria, orderMap, null);
        criteria.setResultTransformer(CriteriaSpecification.ROOT ENTITY);
        return pagedQuery(criteria, pageNo, pageSize);
    }
    @SuppressWarnings("unchecked")
    public Page findBy(Map filterMap, Map orderMap, int pageNo, int pageSize) {
        return findBy(filterMap, orderMap, pageNo, pageSize,
                getDefaultCriteriaSetup());
    }
    public PageInfo<T> pagedQueryForEC(HttpServletRequest request) {
        return pagedQueryForEC(request, null);
    }
    @SuppressWarnings("unchecked")
    public PageInfo<T> pagedQueryForEC(HttpServletRequest request, Map filter) {
        return pagedQueryForEC(request, filter, null);
    }
    @SuppressWarnings("unchecked")
    public PageInfo<T> pagedQueryForEC(HttpServletRequest request, Map filter,
            Integer pageSize) {
        Limit limit - ExtremeTablePage.getLimit(request);
```

```
        int pageNo = limit.getPage();
        if (null == pageSize) {
            pageSize = Constants.DEFAULT PAGE SIZE;
        }
        String ec crd = request
                .getParameter(TableConstants.CURRENT ROWS DISPLAYED);
        Map  filter = ExtremeTablePage.getFilter(limit);
        if (null != filter) {
             filter.putAll(filter);
        }
        if (!StringUtils.isEmpty(ec crd)) {
            pageSize = Integer.parseInt(ec crd);
        }
        Page page = findBy( filter, ExtremeTablePage.getSort(limit), pageNo,
                pageSize);
        PageInfo<T> pageInfo = new PageInfo<T>();
        pageInfo.setCurrentPage(Long.valueOf(page.getCurrentPageNo()));
        pageInfo.setCountOfCurrentPage(Long.valueOf(page.getPageSize()));
        pageInfo.setTotalCount(page.getTotalCount());
        pageInfo.setPageResults((List<T>) page.getResult());
        return pageInfo;
    }

    public PageInfo<T> pagedQueryForEC(HttpServletRequest request,
            Criteria criteria, Map filter, Integer pageSize) {
        Limit limit = ExtremeTablePage.getLimit(request);
        int pageNo = limit.getPage();
        if (null == pageSize) {
            pageSize = Constants.DEFAULT PAGE SIZE;
        }
        String ec crd = request
                .getParameter(TableConstants.CURRENT ROWS DISPLAYED);
        Map  filter = ExtremeTablePage.getFilter(limit);
        if (null != filter) {
             filter.putAll(filter);
        }
        if (!StringUtils.isEmpty(ec crd)) {
            pageSize = Integer.parseInt(ec crd);
        }
        Page page = findBy( filter, ExtremeTablePage.getSort(limit), pageNo,
                pageSize, criteria);
        PageInfo<T> pageInfo = new PageInfo<T>();
        pageInfo.setCurrentPage(Long.valueOf(page.getCurrentPageNo()));
        pageInfo.setCountOfCurrentPage(Long.valueOf(page.getPageSize()));
        pageInfo.setTotalCount(page.getTotalCount());
```

```
        pageInfo.setPageResults((List<T>) page.getResult());
        return pageInfo;
    }
    //Criteria 分页查询，默认 count 模式
     public Page pagedQuery(Criteria criteria, int pageNo, int pageSize) {
        return pagedQuery(criteria, pageNo, pageSize, COUNT MODE);
    }

     //Criteria 分页查询，可以指定 jdbc 是否支持 scroll
    public Page pagedQuery(Criteria criteria, int pageNo, int pageSize, int mode) {
        return CriteriaPage.getPageInstance(criteria, pageNo, pageSize, mode);
    }
     //HQL 分页查询，默认 count 的方式

    public Page pagedQuery(String hql, int pageNo, int pageSize, Object... args) {
        return pagedQuery(hql, pageNo, pageSize, COUNT MODE, args);
    }
    @SuppressWarnings("unchecked")
    public Page pagedQuery(String hql, int pageNo, int pageSize, int mode,
            Object... args) {
        Assert.hasText(hql);
        Query query = getSession().createQuery(hql);
        for (int i = 0; args!=null && i < args.length ; i++) {
            query.setParameter(i, args[i]);
        }
        if (mode == COUNT MODE) {
            String countQueryString = " select count (*) "
                    + removeSelect(removeOrders(hql));
            List countlist = getHibernateTemplate()
                    .find(countQueryString, args);
            Object count = countlist.get(0);
            int totalCount = 0;
            if (count instanceof java.lang.Long)
                totalCount = Integer.parseInt(countlist.get(0).toString());
            if (count instanceof java.lang.Integer)
                totalCount = (Integer) countlist.get(0);
            return HqlPage.getPageInstanceByCount(query, pageNo, pageSize,
                    totalCount);
        } else
            return HqlPage.getPageInstance(query, pageNo, pageSize, mode);
    }
    public void pagedQuery(String hql, PageInfo<T> pageInfo) {
        pagedQuery(hql,pageInfo,null);
    }
    @SuppressWarnings("unchecked")
```

```
    public void pagedQuery(String hql, PageInfo<T> pageInfo, Object... args) {
        Assert.hasText(hql);
        Query query = getSession().createQuery(hql).setCacheable(true);
        if (args != null && args.length != 0) {
            for (int i = 0; i < args.length; i++) {
                query.setParameter(i, args[i]);
            }
        }
        String countQueryString = "select count (*) "
                + removeSelect(removeOrders(hql));
        int totalCount = Integer.valueOf(getHibernateTemplate().find(
                countQueryString, args).get(0).toString());
        pageInfo.setTotalCount(totalCount);
        query.setMaxResults(pageInfo.getCountOfCurrentPage().intValue());
        query.setFirstResult(pageInfo.getCountOfCurrentPage().intValue()
                * (pageInfo.getCurrentPage().intValue() - 1));
        pageInfo.setPageResults(query.list());
    }

    @SuppressWarnings("unchecked")
    public void pagedQueryByQbc(PageInfo<T> pageInfo, Criteria criteria) {
        CriteriaImpl impl = (CriteriaImpl) criteria;
        Projection projection = impl.getProjection();
        List<CriteriaImpl.OrderEntry> orderEntries;
        try {
            orderEntries = (List) BeanUtils.forceGetProperty(impl,
                    "orderEntries");
            BeanUtils.forceSetProperty(impl, "orderEntries", new ArrayList());
        } catch (Exception e) {
            throw new InternalError(" Runtime Exception impossibility throw ");
        }
        int totalCount = (Integer) criteria.setProjection(
                Projections.rowCount()).uniqueResult();
        pageInfo.setTotalCount(totalCount);
        criteria.setProjection(projection);
        if (projection == null) {
            criteria.setResultTransformer(CriteriaSpecification.DISTINCT
            ROOT ENTITY);
        }
        try {
            BeanUtils.forceSetProperty(impl, "orderEntries", orderEntries);
        } catch (Exception e) {
            throw new InternalError(" Runtime Exception impossibility throw ");
        }
        criteria.setCacheable(true);
```

```
        List list = criteria.setFirstResult(
                pageInfo.getCountOfCurrentPage().intValue()
                        * (pageInfo.getCurrentPage().intValue() - 1)).
                        setMaxResults(
                pageInfo.getCountOfCurrentPage().intValue()).list();
        pageInfo.setPageResults(list);
    }
    /**
     * ORACLE 存储过程调用
     *
     * @param hql
     * @param pageInfo
     * @param args
     * @return
     */
    @SuppressWarnings("unchecked")
    public void pagedQueryByOracle(PageInfo pageInfo, LinkedHashMap map) {
        int j = 2;
        Class entityClass = GenericsUtils.getGenericClass(getClass());
        FunctionName anno = (FunctionName) entityClass
                .getAnnotation(FunctionName.class);
        Session session = getSession();
        Query query = session.getNamedQuery(anno.name());
        /** 页数目*/
        query.setParameter(0, pageInfo.getCurrentPage());
        /** 页大小*/
        query.setParameter(1, pageInfo.getCountOfCurrentPage());
        /** 填充数据 */
        for (Iterator ie = map.keySet().iterator(); ie.hasNext();) {
            Object obj = ie.next();
            Object newObject;
            if (map.get(obj.toString()) == null
                    || map.get(obj.toString()).toString().equals("null")) {
                newObject = "";
                query.setParameter(j, newObject);
            } else {
                newObject = map.get(obj.toString());
                query.setParameter(j, newObject);
            }
            j++;
        }
        List list = query.list();
        if (list != null && list.size() != 0) {
            T returnObj = (T) list.get(0);
            String countName = anno.countName();
```

```
            try {
                Method method = returnObj.getClass().getDeclaredMethod(
                        "get" + StringUtils.capitalize(countName));
                try {
                    Object obj = method.invoke(returnObj);
                    pageInfo.setTotalCount(Long.valueOf(obj.toString()));
                } catch (IllegalArgumentException e) {
                    e.printStackTrace();
                } catch (IllegalAccessException e) {
                    e.printStackTrace();
                } catch (InvocationTargetException e) {
                    e.printStackTrace();
                }
            } catch (SecurityException e) {
                e.printStackTrace();
            } catch (NoSuchMethodException e) {
                e.printStackTrace();
            }
        }
        pageInfo.setPageResults(list);
    }
    /**
     * 查询总记录数
     * @param countString
     * @return
     * @throws DataAccessException
     */
    public int count(final String countString) throws DataAccessException {
        return Integer.valueOf(String.valueOf((getHibernate- Template().
        find(
                countString).get(0))));
    }

    private static String removeSelect(String hql) {
        Assert.hasText(hql);
        int beginPos = hql.toLowerCase().indexOf("from");
        Assert.isTrue(beginPos != -1, " hql : " + hql
                + " must has a keyword 'from'");
        return hql.substring(beginPos);
    }

    private static String removeOrders(String hql) {
        Assert.hasText(hql);
        Pattern p = Pattern.compile("order\\s*by[\\w|\\W|\\s|\\S]*",
                Pattern.CASE_INSENSITIVE);
```

```
            Matcher m = p.matcher(hql);
            StringBuffer sb = new StringBuffer();
            while (m.find()) {
                m.appendReplacement(sb, "");
            }
            m.appendTail(sb);
            return sb.toString();
        }
        public List<T> getAllValid() {
            return null;
        }
        public String getUnDeletableHQL() {
            return null;
        }
        public Criterion getUnDeletableCriterion() {
            return null;
        }
    }
```

8.3.2.2 实例的 Web 服务实现

现在继续编写一个 Web 服务接口，以提供外部访问，程序代码如下：

```
    package org.demo.webservice;
    import javax.jws.WebParam;
    import javax.jws.WebService;
    import javax.jws.soap.SOAPBinding;
    import javax.jws.soap.SOAPBinding.Style;
    import javax.jws.soap.SOAPBinding.Use;
    @WebService(name = "CustomerService", serviceName = "CustomerService",
    portName = "CustomerServicePort")
    @SOAPBinding(style=Style.RPC, use=Use.LITERAL)
    public interface ICustomerService {
        /**
         * 得到 customer 的 list 集合，以 json 格式返回
         * @param startNo 开始的页号
         * @param pageSize 每页的 size 大小
         * @return json 格式的 customer 集合
         */
        public String obtainCustomers(@WebParam(name="startNo")int startNo,
        @WebParam(name="pageSize")int pageSize);
        /**
         * 根据 cusomter 的 id 得到一个 Customer 的信息
         * @param id
         * @return 以 json 格式封转
```

```
     */
    public  String obtainCustomerById(@WebParam(name "id")int id);
}
```

下面通过 JSON 实现 Web 服务，以便将业务逻辑发布出去，即发布成 WSDL，详见代码清单 8-3。其他与之相关的代码请详见光盘代码。

代码清单 8-3　通过@WebService 注释将 webservice 发布出去：

```
package org.demo.webservice;
import net.sf.json.JSONObject;
import org.demo.manager.CustomerManager;
import org.demo.model.Customer;
import org.demo.web.CustomerDTO;
import org.demo.web.CustomerList;
import org.framework.core.page.PageInfo;
public class CustomerServiceImpl implements ICustomerService {
    private CustomerManager customerManager;
    public String obtainCustomers(int startNo, int pageSize) {
        PageInfo<Customer> pageInfo = new PageInfo<Customer>();
        pageInfo.setStartNo(Long.valueOf(startNo));
        pageInfo.setCountOfCurrentPage(Long.valueOf(pageSize));
        customerManager.pagedQuery("from Customer", pageInfo);
        CustomerList result =
            new CustomerList(pageInfo.getTotalCount(), pageInfo.
            getPageResults());
                JSONObject jsonResult = JSONObject.fromObject(result);
        return jsonResult.toString();
    }
    public String obtainCustomerById(int id) {
        Customer customer = customerManager.get(Long.valueOf(id));
        CustomerDTO customerDTO = new CustomerDTO(customer.getId(),
        customer.getName(),customer.getEmail(),customer.getBirthday(),
        customer.getMusicCheckBoxs(),customer.getColor(),customer. getFruit(),
        customer.getDescription());
        String result = JSONObject.fromObject(customerDTO).toString();
        return result;
    }
    public void setCustomerManager(CustomerManager customerManager) {
        this.customerManager = customerManager;
    }
}
```

最后配置 Web 服务，并启动后，得到如图 8-7 所示的 Web 服务发布界面，即 WSDL。

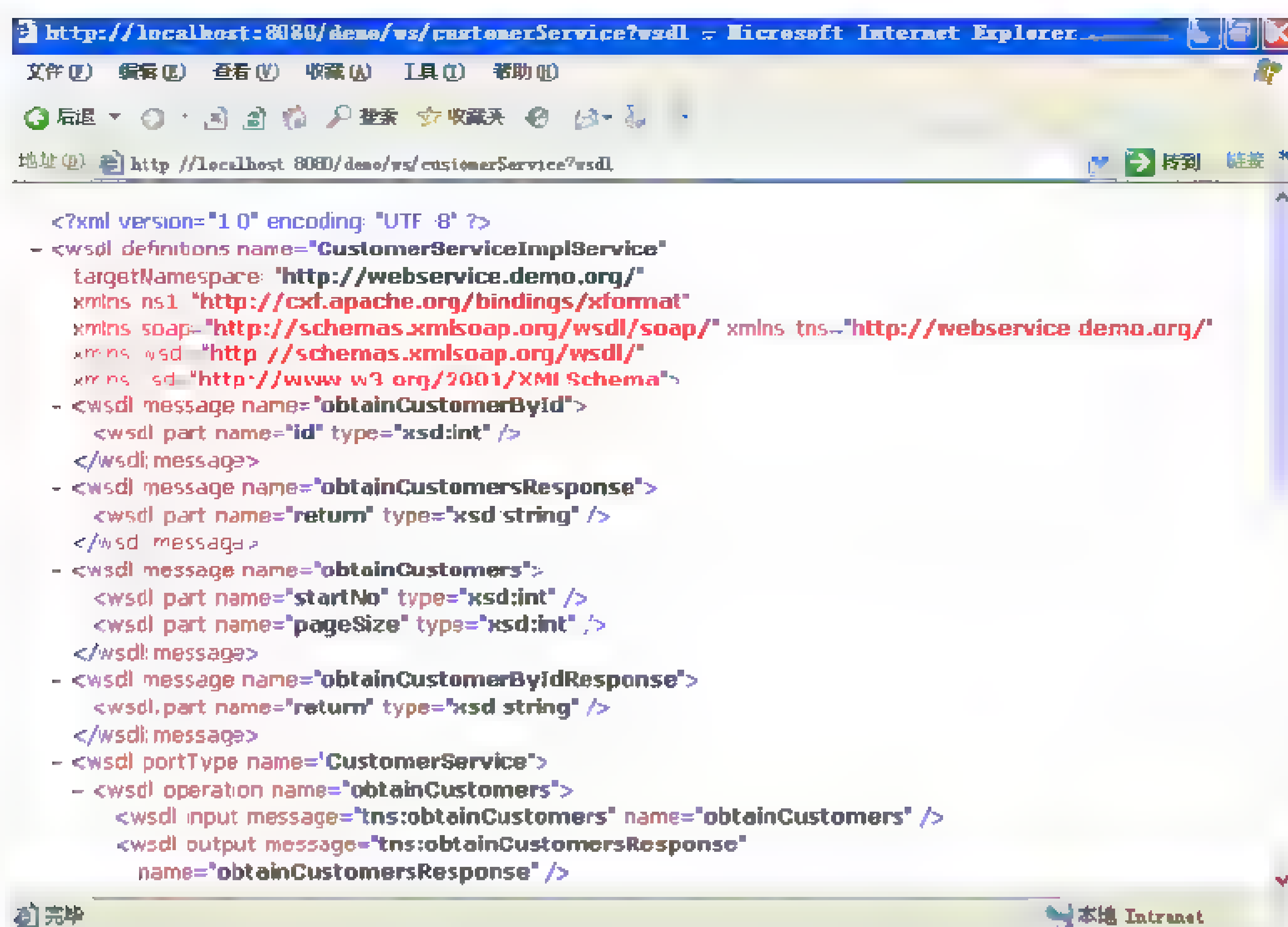

图 8-7　Web 服务发布界面

为可以查看、添加、编辑、删除 Customer 页面显示内容，本书使用 ext.Ext 来与 JSON 格式实现交互，这时需要将 action 都返回 JSON 格式的数据方可实现，如代码清单 8-4。

代码清单 8-4　StrutsAction.java

```java
package org.demo.web;
import java.io.IOException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;
import org.demo.manager.CustomerManager;
import org.demo.model.Color;
import org.demo.model.Customer;
import org.demo.model.Fruit;
import org.demo.webservice.ICustomerService;
import org.framework.core.commons.ReturnType;
import org.framework.core.controller.StrutsEntityAction;
import org.springframework.util.StringUtils;
import org.springframework.web.bind.ServletRequestBindingException;
import org.springframework.web.bind.ServletRequestUtils;

public class CustomerAction extends StrutsEntityAction<Customer,
CustomerManager> {
    @SuppressWarnings("unused")
    private CustomerManager customerManager;
    private ICustomerService customerService;
    /**
     * 以 json 格式返回所有 Customer、Ext 需要解析 json 格式数据
```

```
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @throws IOException
 */
public void obtainCustomers(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response)
    throws IOException {
       response.setContentType( "text/html;charset=UTF-8 ");
       response.setCharacterEncoding( "UTF-8");
       int start;
    try {
        start = ServletRequestUtils.getIntParameter(request, "start");
      }
    catch (ServletRequestBindingException e) {
        log.info("start paramter is empty, default is 1");
        start =1;
    }
    log.info("start:"+start);
            int limit = ServletRequestUtils.getIntParameter(request,
            "limit", 10);
    log.info("limit:"+limit);
    String result = customerService.obtainCustomers(start, limit);
    String header = ServletRequestUtils.getStringParameter(request,
    "callback", null);
    if (StringUtils.hasText(header)) {
        result = header + "(" + result + ")";
    }
    response.getWriter().print(result);
    }
/**
 * 根据 customer id，得到 json 格式的 customer 信息
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @throws IOException
 */
public void obtainCustomerById(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response)
    throws IOException {
    response.setContentType( "text/html;charset-UTF-8 ");
    response.setCharacterEncoding( "UTF-8");
    int id - ServletRequestUtils.getIntParameter(request, "id", 0);
```

```
        if (id!=0) {
            String result = customerService.obtainCustomerById(id);
            response.getWriter().print(wrapSuccessData(result));
        }
    }
    /**
     * 在初始化 form 表单时，设置 music,color,fruit 信息，初始化操作，都可以写在这里
     */
    @Override
    protected void onInitForm(ActionForm form, HttpServletRequest request,
            Customer customer) {
        request.setAttribute("musics", customer.getMusicCheckBoxs());
        request.setAttribute("colors", Color.values());
        request.setAttribute("fruits", Fruit.values());
    }
    /**
     * 采用的是 json 格式，所以复写此方法，表明是 json,如果是常规方式，不用复写
     */
    @Override
    protected ReturnType returnType() {
        return ReturnType.JSON;
    }
    public void setCustomerManager(CustomerManager customerManager) {
        this.customerManager = customerManager;
    }
    public void setCustomerService(ICustomerService customerService) {
        this.customerService = customerService;
    }
}
```

现在就需要对 StrutsAction 在 WebRoob\Web-INF\struts-config.xml 进行配置，其配置 xml 见前面。但需要对以下代码进一步解释：

```
<action path="/customer" name="customerForm"
scope="request" parameter="method" validate="false">
            <forward name="list"
path="customer.do?method=direct" redirect="true"/>
            <forward name="direct"
path="/WEB-INF/pages/customer/customerList.jsp"/>
            <forward name="edit"
path="/WEB-INF/pages/customer/customerEdit.jsp"/>
            <forward name="success"
path="customer.do?method=direct" redirect="true"/>
</action>
```

在程序代码中：

List 表示会把所有的 Entity 传入页面；Direct 表示直接进入页面；Edit 表示当需要编辑一个实体时，即得到相应 Entity 信息，然后进入对应编辑页面；Save 表示当需要添加一个实体时，得到相应 Entity 信息，然后进入对应添加页面；

由于实例使用开源软件 Ext（相关 Ext 的叙述见 5.5.2.4 小节），若需要让 Ext 去自动加载所有信息，则首先需要配置 Direct，这个 direct 将作为首页面。同时把 List 也指向 direct 的 action，即使访问 list 时，其实也是访问 direct。

这时还需要继续在 WebRoot\Web-INF\action-servlet.xml 下配置 bean，即添加<bean name="/customer" class="org.demo.web.CustomerAction" />与 spring 关联，具体详细配置可以查看前面的配置文件。

8.3.2.3　实例 JSP 页面

JSP 里里面主要编写了实现开源软件 EXT 的方法，由于 EXT 在处理逻辑上显示时，是采用 JS 来实现的，而往往业务逻辑是 bean。因此，EXT 采用 JSON 来完成业务逻辑“读取”服务并显示（由于这些内容实现是比较新颖的，将其中关键的代码全列举出来），其运行结果如图 8-8 所示。则客户信息与偏好显示 JSP 页面（customer_list.jsp）如代码清单 8-5 所示。

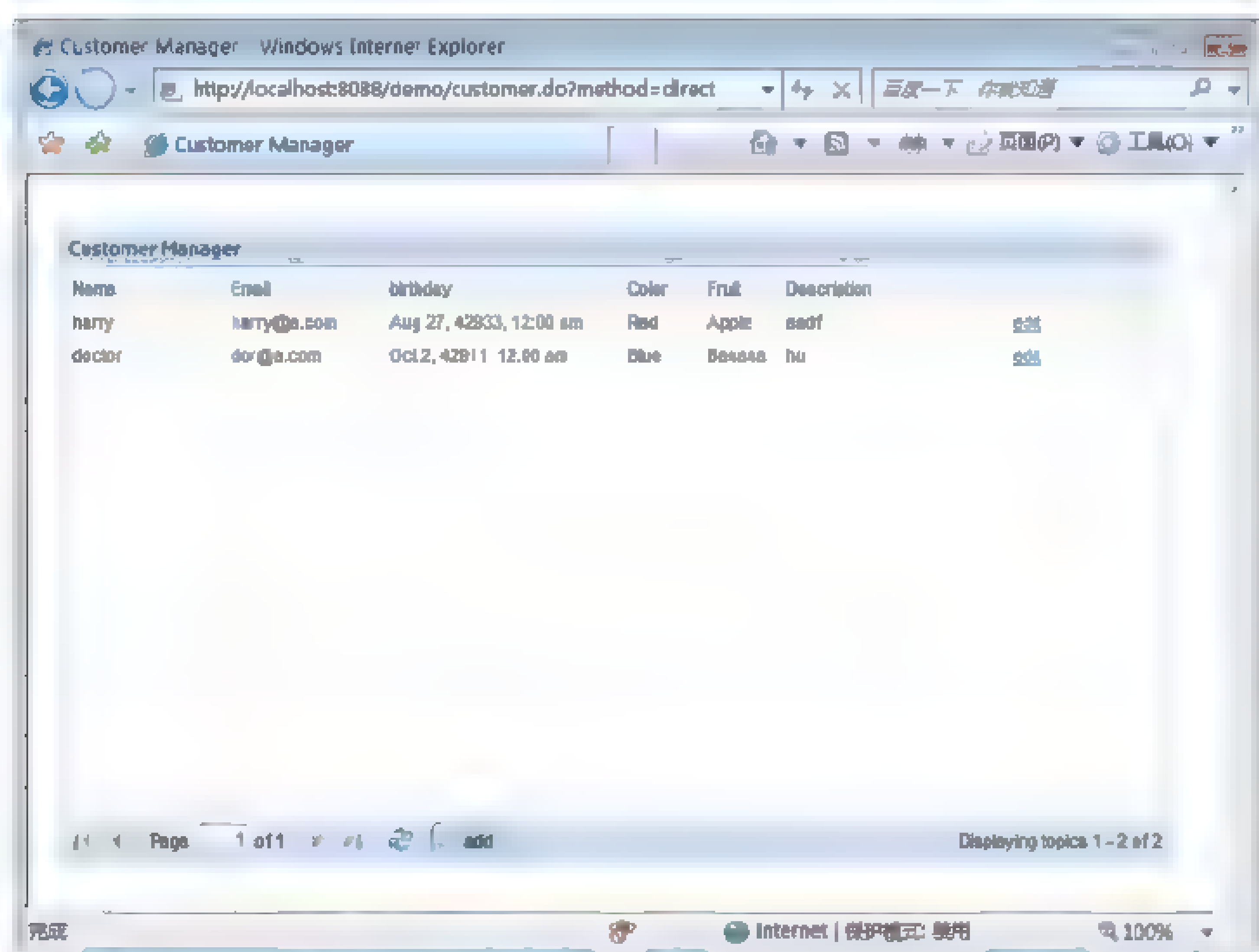

图 8-8　customer_list.jsp 运行结果

代码清单 8-5　Customer_list.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf8">
<title>Customer Manager</title>
    <link rel="stylesheet" type="text/css"
        href="${ctx }/scripts/ext-3.3.0/resources/css/ext-all.css" />
```

```
        <script type "text/javascript"
        src "${ctx }/scripts/ext-3.3.0/adapter/ext/ext base.js"></script>
        <script type="text/javascript" src="${ctx }/scripts/commonjs.js">
        </script>
    <script type="text/javascript" src="${ctx }/scripts/ext-3.3.0/ext-
    all.js"></script>
    <script type="text/javascript" src="${ctx }/scripts/customer list.
    js"></script>
    <link rel="stylesheet" type="text/css" href="${ctx }/styles/grid.css" />
    <link rel="stylesheet" type="text/css" href="${ctx }/styles/common.css" />
</head>
<body>
<div>
    <label>name:</label><input  id="customer name"  name="customer name">
    <button onclick="search()">查询</button>
</div>
<div id="topic-grid"></div>
</body>
</html>
```

上述代码中的<script type="text/javascript" src="${ctx }/scripts/customer_list.js">中，<script>表示导入显示 customer 列表的业务 JS。其他都是调用 ext 需要的依赖 js 包，其中 customer_list.js 是本实例创建的 JS 文件，还有其他相关的源代码请详见光盘，并且需要进一步了解 EXT 使用方法，还可以登录 http://www.sencha.com/products/js/网站来学习。下面的代码清单 8-6——Customer_list.js 是本实验编写使用的 EXT 代码：

代码清单 8-6 Customer_list.js：

```
var store;
Ext.onReady(function() {
    //创建 Data Store,json 格式
    store = new Ext.data.JsonStore( {
        root : 'customers',             //json 数据格式的根的名字
        totalProperty : 'totalCount',   //需要的总记录数
        idProperty : 'id',              //id 的名称
        remoteSort : true,              //是否是远程调用
        //字段的定义,
        fields : [
        'name', 'email',
        //下面 birthday 是日期的，所以复杂些
        {
            name : 'birthday',
            mapping : 'birthday',
            type : 'date',
            dateFormat : 'timestamp'
        },'color','fruit','description'
```

```
        ],
        //访问 action 得到 json 格式的 cutomer 信息
        proxy : new Ext.data.HttpProxy( {
            url : 'customer.do?method=obtainCustomers'
        })
    });
    function renderLast(value, p, r) {
        return value.dateFormat('M j, Y, g:i a');
    }
    //格式化一段 String 返回，是一个编辑的链接
    function renderAddButton(value, p, r) {
        return String.format("<a href='customer.do?method=edit&id={0}'>
        edit</a>",r.id);
    }
    //Ext 表单的配置
    var grid = new Ext.grid.GridPanel( {
        width : 700,                    //宽
        height : 400,                   //长
        title : 'Customer Manager',     //title
        store : store,//store 就是上面刚定义的 store
        trackMouseOver : false,
        disableSelection : true,
        loadMask : true,
        //表单的具体列有哪些
        columns : [ {
            id : 'topic',               //id 的 css 样式名，只是为了 css 使用
            header : "Name",            //header 显示名称
            dataIndex : 'name',         //在 js 的数据中存储的索引名
            width : 100,//列宽
            //renderer: renderTopic,
            sortable : true
        }, {
            header : "Email",
            dataIndex : 'email',
            width : 100,
            hidden : false,
            sortable : true
        }, {
            id : 'last',
            header : "birthday",
            dataIndex : 'birthday',
            width : 150,
            renderer : renderLast, //日期使用函数返回的 String 显示，自定义
            sortable : true
```

```
        }, {
            header : "Color",
            dataIndex : 'color',
            width : 50,
            hidden : false,
            sortable : true
        }, {
            header : "Fruit",
            dataIndex : 'fruit',
            width : 50,
            hidden : false,
            sortable : true
        }, {
            header : "Description",
            dataIndex : 'description',
            width : 100,
            hidden : false,
            sortable : true
        }, {
            width : 70,
            align : 'right',
            renderer : renderAddButton
        }       ],
        //分页工具
        bbar : new Ext.PagingToolbar( {
            pageSize : 10, //每页 10 条
            store : store,//store 仓库为先前定义的 store
            displayInfo : true,
            displayMsg : 'Displaying topics {0} - {1} of {2}',//显示格式
            emptyMsg : "No topics to display"       ,
         //添加一个 add 按钮，用以添加先得 Customer
         items:[
         '-', {
         pressed: true,
         enableToggle:true,
         text: 'add',
         cls: 'x btn text icon',
         toggleHandler: function(btn, pressed){
             location.href = "customer.do?method=create";
         }
         }]
        })
```

```
        });
        //开始绘制页面
        grid.render('topic grid');
        //开始读取数据
        store.load( {
            params : {
                start : 0,
                limit : 10
            }
        });
    });
    function search(){
        store.load( {
            params : {
                start : 0,
                limit : 10,
                name: Ext.get('customer_name').dom.value
            }
        });
    }
```

其 customerEdit.jsp 页面实现方法与 customer_list.jsp 实现方法相同，这里就不再详细介绍了，具体可以参见光盘源代码。这时当点击图 8-8 中的“edit”时，可得到图 8-9 所示的界面；当单击图 8-8 中的“add”时，可以得到图 8-10 所示的界面。

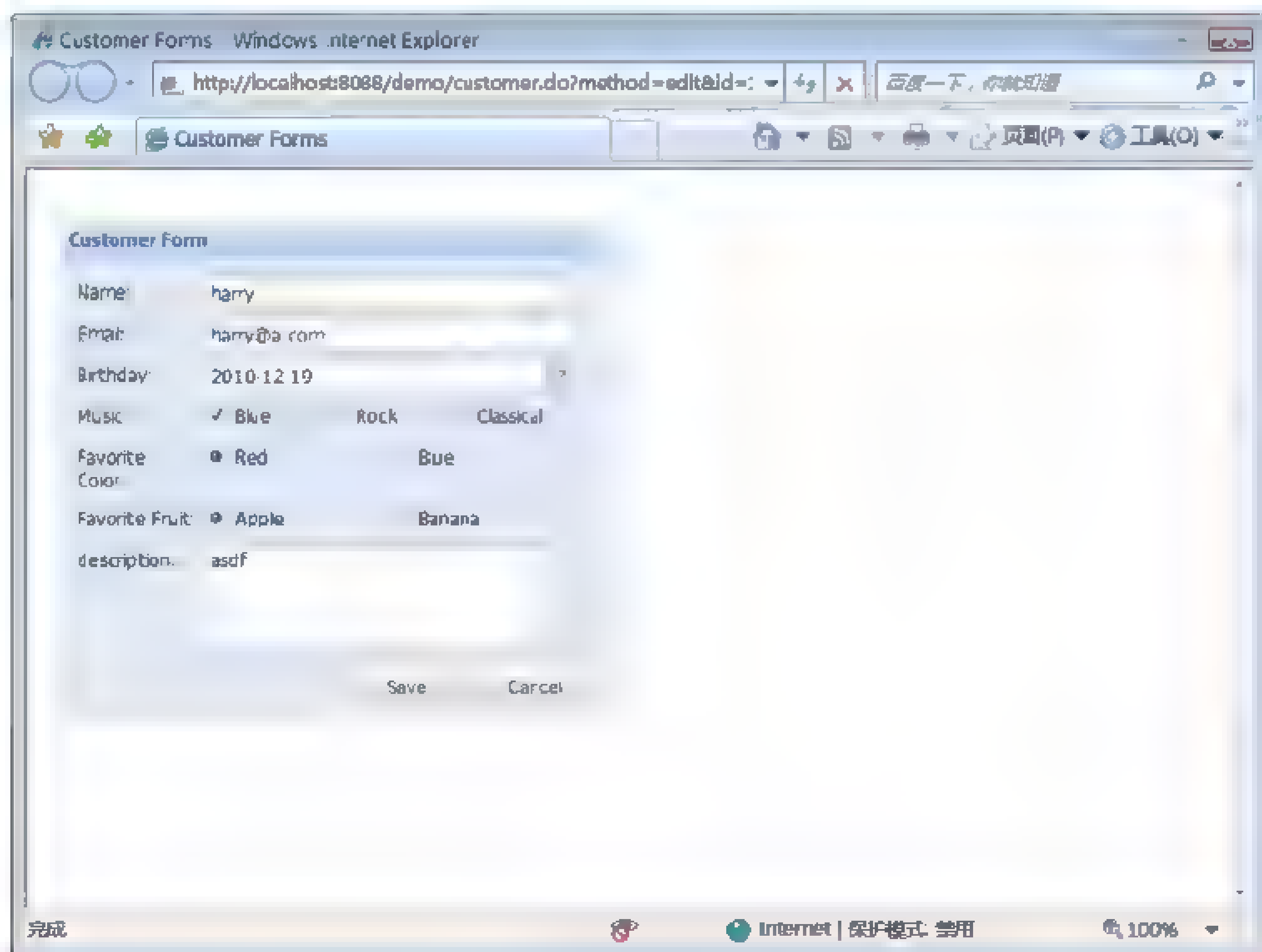

图 8-9　应用实例的编辑界面

至此，应用实例的服务器描述就完成了，接下来就是实现客户端了。

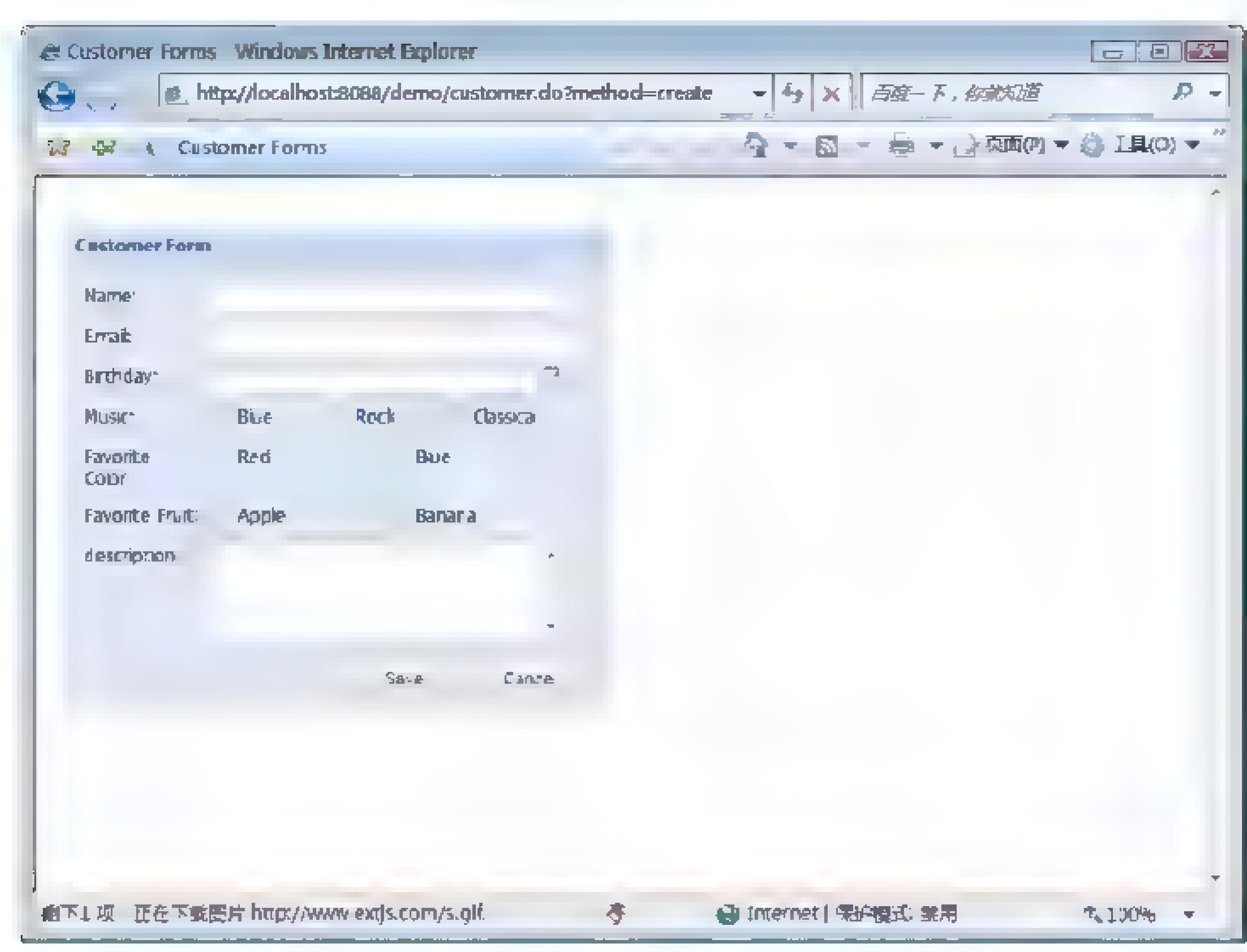

图 8-10　应用实例的增加界面

8.3.2.4　应用举例的客户端

这时，将图 8-7 中的结构 http://localhost:8080/demo/ws/customerService?wsdl 结果放入到图 8-5 所示的 WSDL2OWL-S 转换页面中，得到图 8-11 所示的结果。

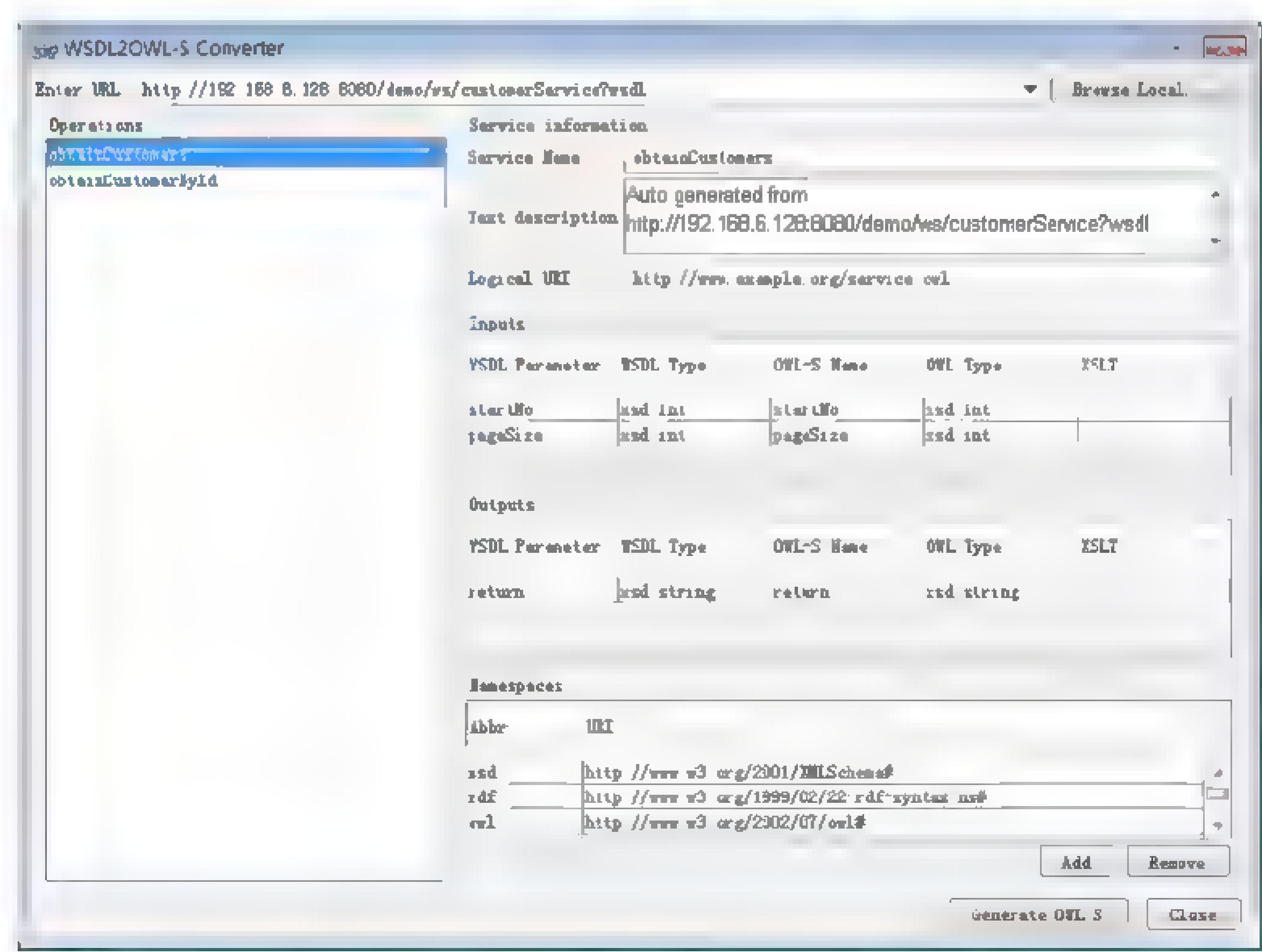

图 8-11　基于 WSDL2OWL-S 的语义转换

通过 WSDL2OWL-S 转换可以获得两个 OWL-S 文件，分别如下：

（1）获得客户信息：obtainCustomers.owl。

```
<?xml version "1.0"?>
```

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:expr="http://www.daml.org/services/owl-s/1.2/generic/Expression.
    owl#"
    xmlns:service="http://www.daml.org/services/owl-s/1.2/Service.owl#"
    xmlns:process="http://www.daml.org/services/owl-s/1.2/Process.owl#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.example.org/service.owl"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:swrl="http://www.w3.org/2003/11/swrl#"
    xmlns:grounding="http://www.daml.org/services/owl-s/1.2/Grounding.
    owl#"
    xmlns:profile="http://www.daml.org/services/owl-s/1.2/Profile.owl#"
    xmlns:list="http://www.daml.org/services/owl-s/1.2/generic/ObjectList.
    owl#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://www.example.org/service.owl">
  <owl:Ontology rdf:about="">
    <owl:imports
      rdf:resource="http://www.daml.org/services/owl-s/1.2/Grounding.owl"/>
    <owl:imports
      rdf:resource="http://www.daml.org/services/owl-s/1.2/Profile.owl"/>
  </owl:Ontology>
  <service:Service rdf:ID="obtainCustomerByIdService">
    <service:supports>
      <grounding:WsdlGrounding rdf:ID="obtainCustomerByIdGrounding"/>
    </service:supports>
    <service:describedBy>
      <process:AtomicProcess rdf:ID="obtainCustomerByIdProcess"/>
    </service:describedBy>
    <service:presents>
      <profile:Profile rdf:ID="obtainCustomerByIdProfile"/>
    </service:presents>
  </service:Service>
  <profile:Profile rdf:about="#obtainCustomerByIdProfile">
    <profile:hasOutput>
      <process:Output rdf:ID="return">
        <process:parameterType
        rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
        http://www.w3.org/2001/XMLSchema#string</process:parameterType>
        <rdfs:label>return</rdfs:label>
      </process:Output>
    </profile:hasOutput>
    <profile:hasInput>
      <process:Input rdf:ID="id">
```

```
        <process:parameterType
        rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
        http://www.w3.org/2001/XMLSchema#int</process:parameterType>
        <rdfs:label>id</rdfs:label>
      </process:Input>
    </profile:hasInput>
    <profile:textDescription>Auto generated from
  http://192.168.222.128:8008/demo/ws/customerService?wsdl
  </profile:textDescription>
    <profile:serviceName>obtainCustomerById</profile:serviceName>
    <service:presentedBy rdf:resource="#obtainCustomerByIdService"/>
  </profile:Profile>
  <process:AtomicProcess rdf:about="#obtainCustomerByIdProcess">
    <process:hasOutput rdf:resource="#return"/>
    <process:hasInput rdf:resource="#id"/>
    <service:describes rdf:resource="#obtainCustomerByIdService"/>
    <rdfs:label>obtainCustomerByIdProcess</rdfs:label>
  </process:AtomicProcess>
  <grounding:WsdlGrounding rdf:about="#obtainCustomerByIdGrounding">
    <grounding:hasAtomicProcessGrounding>
      <grounding:WsdlAtomicProcessGrounding
      rdf:ID="obtainCustomerByIdAtomicProcessGrounding"/>
    </grounding:hasAtomicProcessGrounding>
    <service:supportedBy rdf:resource="#obtainCustomerByIdService"/>
  </grounding:WsdlGrounding>
  <grounding:WsdlAtomicProcessGrounding
    rdf:about="#obtainCustomerByIdAtomicProcessGrounding">
    <grounding:wsdlOutput>
      <grounding:WsdlOutputMessageMap>
        <grounding:wsdlMessagePart
    rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
  http://192.168.222.128:8008/demo/ws/customerService?wsdl#return
</grounding:wsdlMessagePart>
        <grounding:owlsParameter rdf:resource="#return"/>
      </grounding:WsdlOutputMessageMap>
    </grounding:wsdlOutput>
    <grounding:wsdlInput>
      <grounding:WsdlInputMessageMap>
        <grounding:wsdlMessagePart rdf:datatype="http://www.w3.org/2001/
        XMLSchema#anyURI">
http://192.168.222.128:8008/demo/ws/customerService?wsdl#id
</grounding:wsdlMessagePart>
        <grounding:owlsParameter rdf:resource-"#id"/>
      </grounding:WsdlInputMessageMap>
    </grounding:wsdlInput>
```

```
    <grounding:wsdlOutputMessage rdf:datatype "http://www.w3.org/2001/
    XMLSchema#anyURI">
http://webservice.demo.org/#obtainCustomerByIdResponse
</grounding:wsdlOutputMessage>
    <grounding:wsdlInputMessage
    rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
    http://webservice.demo.org/#obtainCustomerById
</grounding:wsdlInputMessage>
    <grounding:wsdlDocument
    rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://192.168.222.128:8008/demo/ws/customerService?wsd
</grounding:wsdlDocument>
    <grounding:wsdlOperation>
      <grounding:WsdlOperationRef>
        <grounding:operation
        rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
      http://192.168.222.128:8008/demo/ws/customerService?wsdl#
       obtainCustomerById
</grounding:operation>
      </grounding:WsdlOperationRef>
    </grounding:wsdlOperation>
    <grounding:owlsProcess rdf:resource="#obtainCustomerByIdProcess"/>
  </grounding:WsdlAtomicProcessGrounding>
</rdf:RDF>
```

（2）获取每一个客户信息与偏好：obtainCustomerById.owl。

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:expr="http://www.daml.org/services/owl-s/1.2/generic/
    Expression.owl#"
    xmlns:service="http://www.daml.org/services/owl-s/1.2/Service.owl#"
    xmlns:process="http://www.daml.org/services/owl-s/1.2/Process.owl#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.example.org/service.owl"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:swrl "http://www.w3.org/2003/11/swrl#"
    xmlns:grounding="http://www.daml.org/services/owl-s/1.2/
    Grounding.owl#"
    xmlns:profile "http://www.daml.org/services/owl s/1.2/Profile.owl#"
    xmlns:list="http://www.daml.org/services/owl-s/1.2/generic/ ObjectList.
    owl#"
    xmlns:rdfs-"http://www.w3.org/2000/01/rdf schema#"
  xml:base "http://www.example.org/service.owl">
  <owl:Ontology rdf:about "">
```

```
    <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.2/
    Grounding.owl"/>
    <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.2/
    Profile.owl"/>
  </owl:Ontology>
  <service:Service rdf:ID="obtainCustomerByIdService">
    <service:supports>
      <grounding:WsdlGrounding rdf:ID="obtainCustomerByIdGrounding"/>
    </service:supports>
    <service:describedBy>
      <process:AtomicProcess rdf:ID="obtainCustomerByIdProcess"/>
    </service:describedBy>
    <service:presents>
      <profile:Profile rdf:ID="obtainCustomerByIdProfile"/>
    </service:presents>
  </service:Service>
  <profile:Profile rdf:about="#obtainCustomerByIdProfile">
    <profile:hasOutput>
      <process:Output rdf:ID="return">
        <process:parameterType rdf:datatype="http://www.w3.org/2001/
        XMLSchema#anyURI"
        >http://www.w3.org/2001/XMLSchema#string</process:parameterType>
        <rdfs:label>return</rdfs:label>
      </process:Output>
    </profile:hasOutput>
    <profile:hasInput>
      <process:Input rdf:ID="id">
        <process:parameterType rdf:datatype="http://www.w3.org/2001/
        XMLSchema#anyURI"
        >http://www.w3.org/2001/XMLSchema#int</process:parameterType>
        <rdfs:label>id</rdfs:label>
      </process:Input>
    </profile:hasInput>
    <profile:textDescription>
Auto generated from
http://192.168.222.128:8008/demo/ws/customerService?wsdl
</profile:textDescription>
    <profile:serviceName>obtainCustomerById</profile:serviceName>
    <service:presentedBy rdf:resource="#obtainCustomerByIdService"/>
  </profile:Profile>
  <process:AtomicProcess rdf:about="#obtainCustomerByIdProcess">
    <process:hasOutput rdf:resource="#return"/>
    <process:hasInput rdf:resource="#id"/>
    <service:describes rdf:resource="#obtainCustomerByIdService"/>
    <rdfs:label>obtainCustomerByIdProcess</rdfs:label>
```

```
    </process:AtomicProcess>
    <grounding:WsdlGrounding rdf:about="#obtainCustomerByIdGrounding">
      <grounding:hasAtomicProcessGrounding>
        <grounding:WsdlAtomicProcessGrounding rdf:ID="obtainCustomerBy-
        IdAtomicProcessGrounding"/>
      </grounding:hasAtomicProcessGrounding>
      <service:supportedBy rdf:resource="#obtainCustomerByIdService"/>
    </grounding:WsdlGrounding>
    <grounding:WsdlAtomicProcessGrounding rdf:about="#obtainCustomerBy-
    IdAtomicProcessGrounding">
      <grounding:wsdlOutput>
        <grounding:WsdlOutputMessageMap>
          <grounding:wsdlMessagePart rdf:datatype="http://www.w3.org/2001/
          XMLSchema#anyURI">
  http://192.168.222.128:8008/demo/ws/customerService?wsdl#return
  </grounding:wsdlMessagePart>
          <grounding:owlsParameter rdf:resource="#return"/>
        </grounding:WsdlOutputMessageMap>
      </grounding:wsdlOutput>
      <grounding:wsdlInput>
        <grounding:WsdlInputMessageMap>
          <grounding:wsdlMessagePart rdf:datatype="http://www.w3.org/2001/
          XMLSchema#anyURI">
  http://192.168.222.128:8008/demo/ws/customerService?wsdl#id
  </grounding:wsdlMessagePart>
          <grounding:owlsParameter rdf:resource="#id"/>
        </grounding:WsdlInputMessageMap>
      </grounding:wsdlInput>
      <grounding:wsdlOutputMessage rdf:datatype="http://www.w3.org/2001/
      XMLSchema#anyURI"
      >http://webservice.demo.org/#obtainCustomerByIdResponse</grounding:
      wsdlOutputMessage>
      <grounding:wsdlInputMessage rdf:datatype="http://www.w3.org/2001/
      XMLSchema#anyURI"
      >http://webservice.demo.org/#obtainCustomerById</grounding:
      wsdlInputMessage>
      <grounding:wsdlDocument rdf:datatype="http://www.w3.org/2001/
      XMLSchema#anyURI"
      >http://192.168.222.128:8008/demo/ws/customerService?wsdl</grounding:
      wsdlDocument>
      <grounding:wsdlOperation>
        <grounding:WsdlOperationRef>
          <grounding:operation rdf:datatype="http://www.w3.org/2001/
          XMLSchema#anyURI">
```

```
        http://192.168.222.128:8008/demo/ws/customerService?wsdl#
        obtainCustomerById
</grounding:operation>
      </grounding:WsdlOperationRef>
    </grounding:wsdlOperation>
    <grounding:owlsProcess rdf:resource="#obtainCustomerByIdProcess"/>
  </grounding:WsdlAtomicProcessGrounding>
</rdf:RDF>
```

现在读取这两个 OWL 文件在客户端显示，该类为 CustomerOWLServiceImpl.java。其代码清单 8-7 如下：

代码清单 8-7　CustomerOWLServiceImpl.java：

```
import org.mindswap.exceptions.ExecutionException;
import org.mindswap.owl.OWLDataValue;
import org.mindswap.owl.OWLFactory;
import org.mindswap.owl.OWLKnowledgeBase;
import org.mindswap.owl.OWLValue;
import org.mindswap.owls.OWLSFactory;
import org.mindswap.owls.process.Process;
import org.mindswap.owls.process.execution.ProcessExecutionEngine;
import org.mindswap.owls.process.variable.Input;
import org.mindswap.owls.process.variable.Output;
import org.mindswap.owls.service.Service;
import org.mindswap.query.ValueMap;
public class CustomerOWLServiceImpl implements ICustomerOWLService {
    public String obtainCustomers(int start,int limit) {
        String result = null;
        try {
            //创建一个 engine
            ProcessExecutionEngine exec = OWLSFactory.createExecutionEngine();
            //得到 KnowledgeBase
            OWLKnowledgeBase kb = OWLFactory.createKB();
            kb.setReasoner("RDFS");
            //读取服务
            Service aService;
            //从 owl 读取所有 customer 的信息，obtainCustomers.owl 由 wsdl2owl 生成
            aService =
                        kb.readService(CustomerOWLServiceImpl.class.
                        getClassLoader().
                        getResource("owl/obtainCustomers.owl").toURI());
            //aService = kb.readService(new File("resources/owl/
            obtainCustomers.owl").toURI());
            //从服务得到 process
            Process aProcess = aService.getProcess();
```

```
            //创建 一个参数 map
            ValueMap<Input, OWLValue> inputs = new ValueMap<Input, OWLValue>();
            //设置读取参数
            inputs.setValue(aProcess.getInput("startNo"),
                    kb.createDataValue(start));
            inputs.setValue(aProcess.getInput("pageSize"),
                    kb.createDataValue(limit));
            //执行服务，得到返回结果
            ValueMap<Output, OWLValue> outputs = exec.execute(aProcess,
            inputs, kb);
            OWLDataValue out = outputs.getDataValue(aProcess.getOutput());
            result = out.getValue().toString();
        } catch (IOException e) {
            throw new RuntimeException(e);
        } catch (ExecutionException e) {
            throw new RuntimeException(e);
        } catch (URISyntaxException e) {
            throw new RuntimeException(e);
        }
        return result;
    }

    public String obtainCustomerById(int id) {
        String result = null;
        try {
            //创建一个 engine
            ProcessExecutionEngine exec = OWLSFactory.createExecutionEngine();
            //得到 KnowledgeBase
            OWLKnowledgeBase kb = OWLFactory.createKB();
            kb.setReasoner("RDFS");
            //读取服务
            Service aService;
            //aService = kb.readService(new File("src/obtainCustomerById.
            owl").toURI());
            aService =
                    kb.readService(CustomerOWLServiceImpl.class.
                    getClassLoader().
                    getResource("owl/obtainCustomerById.owl").toURI());
            //aService = kb.readService(new URL("http://abc.owl").toURI());
            //从服务得到 process
            Process aProcess = aService.getProcess();
            //创建一个参数 map
            ValueMap<Input, OWLValue> inputs = new ValueMap<Input, OWLValue>();
            //设置读取参数
            inputs.setValue(aProcess.getInput("id"),
```

```
                    kb.createDataValue(id));
                //执行服务，得到返回结果
                ValueMap<Output, OWLValue> outputs = exec.execute(aProcess,
                inputs, kb);
                OWLDataValue out = outputs.getDataValue(aProcess.getOutput());
                result = out.getValue().toString();
            } catch (IOException e) {
                throw new RuntimeException(e);
            } catch (ExecutionException e) {
                throw new RuntimeException(e);
            } catch (URISyntaxException e) {
                throw new RuntimeException(e);
            }
            return result;
        }
    }
```

通过上述代码就完成了 OWL 读取，现就可以在客户端读取 owl 内容，并显示在页面上。这时所有服务都是服务器端提供，所以当客户端读取内容，就可以直接显示在页面上了。这时 web 层的 Action 代码清单 8-8 如下：

代码清单 8-8　Web 层的 Action 代码：

```
package org.web;
import java.io.IOException;
import java.util.HashSet;
import java.util.Set;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import net.sf.ezmorph.bean.MorphDynaBean;
import net.sf.json.JSONObject;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.actions.DispatchAction;
import org.owl.ICustomerOWLService;
import org.springframework.util.StringUtils;
import org.springframework.web.bind.ServletRequestBindingException;
import org.springframework.web.bind.ServletRequestUtils;
public class CustomerAction extends DispatchAction {
    private ICustomerOWLService customerOWLService;
    public ActionForward list(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response)
            throws IOException {
        return mapping.findForward("list");
    }
```

```
/**
 * 获取所有的 customer 的列表，应为我们使用 EXT 显现，所以在页面上输出 一段 Ext 需要的
 json 格式.
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @throws IOException
 */
public void obtainCustomers(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response)
        throws IOException {
    //在页面显示所以的 customer 列表
    response.setContentType("text/html;charset=UTF-8 ");
    response.setCharacterEncoding("UTF-8");
    int start;
    try {
        start = ServletRequestUtils.getIntParameter(request, "start");
    } catch (ServletRequestBindingException e) {
        log.info("start paramter is empty, default is 1");
        start = 1;
    }
    log.info("start:" + start);
    int limit = ServletRequestUtils.getIntParameter(request, "limit", 10);
    log.info("limit:" + limit);
    String result = customerOWLService.obtainCustomers(start, limit);
    String header = ServletRequestUtils.getStringParameter(request,
            "callback", null);
    if (StringUtils.hasText(header)) {
        result = header + "(" + result + ")";
    }
    response.getWriter().print(result);
}
/**
 * 获取 一个 Customer 的信息，得到 json 格式的数据，然后转化为 javabean，方便后面生成
 表单.
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return
 * @throws IOException
 */
public ActionForward detail(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response)
```

```
            throws IOException {
    //查看具体的第一个Customer，它不具备编辑功能，应为读取的是服务器的信息，这里只是查看
        int id = ServletRequestUtils.getIntParameter(request, "id", 0);
        if (id != 0) {
            String customer = customerOWLService.obtainCustomerById(id);
            log.info("customer:" + customer);
            //使用 JSON-Lib 转化 string 的 json 为一个 javabean
            JSONObject object = JSONObject.fromObject(customer);
            CustomerDTO customerDTO =
                            (CustomerDTO)JSONObject.toBean(object,CustomerDTO.
                            class);
                    //CustomerDTO 里面还包含了 Color、Fruit、Music 和 Music-
                        CheckBox。这些都属于 DTO 的 javabean，用来帮助封转数据格式
                //由于 CustomerDTO 里面有个 Set 集合，里面是复杂对象，JSONObject 会返
                  回一个 MorphDynaBean 对象，所以这里还需要将里面的 Set 集合转化为需要
                  的 MusicCheckBox 对象
            Set musics = customerDTO.getMusic();
            Set<MusicCheckBox>  _musics = new HashSet<MusicCheckBox>();
            for (Object music : musics) {
                MorphDynaBean bean = (MorphDynaBean)music;
                MusicCheckBox checkBox =
                new MusicCheckBox(Music.valueOf(bean.get("music").
              toString()),(Boolean)bean.
              get("checked"));
                _musics.add(checkBox);
            }
            customerDTO.setMusic(_musics);
            //将页面需要的信息传入
            request.setAttribute("customer", customerDTO);
            request.setAttribute("colors", Color.values());
            request.setAttribute("fruits", Fruit.values());
        }
        return mapping.findForward("detail");
    }
    public void setCustomerOWLService(ICustomerOWLService
    customerOWLService) {
        this.customerOWLService = customerOWLService;
    }
}
```

像其他应用一样，还是需要配置 Struts 以实现 OWL 数据显示，与之相关的其他代码请查看光盘。首先配置 bean，以便使用 spring 的特性，代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
"http://www.springframework.org/dtd/spring-beans-2.0.dtd">
```

```
<beans default-autowire "byName" default lazy init "true" >
    <bean name "/customer" class "org.web.CustomerAction" />
</beans>
```

然后配置 struts-config.xml，代码如下。其实这些配置方法与前面讲解的配置方法是一样的。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts-config PUBLIC
        "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
        "http://struts.apache.org/dtds/struts-config 1 2.dtd">
<struts-config>
    <form-beans>
        <form-bean name="customerForm" type="org.apache.struts.validator.
        LazyValidatorForm">
         /*用来映射表单需要的那些属性*/
            <form-property name="name" type="java.lang.String"/>
            <form-property name="email" type="java.lang.String"/>
            <form-property name="birthday" type="java.util.Date"/>
            <form-property name="musics" type="java.lang.String[]"/>
            <form-property name="color" type="java.lang.String"/>
            <form-property name="fruit" type="java.lang.String"/>
            <form-property name="description" type="java.lang.String"/>
            <form-property name="duration" type="java.lang.Long"/>
        </form-bean>
    </form-beans>
    <action-mappings>
            <action path="/customer" name="customerForm" scope="request"
            /*用来配置 action 的 url 访问*/
            parameter="method" validate="false">
            <forward name="list" path="/WEB-INF/pages/customer/
            customerList.jsp"/>
            /*list 用来显示 customer 列表*/
            <forward name="detail" path="/WEB-INF/pages/customer/
            customerEdit.jsp"/>
            /* Name = detail 用来查看具体的一个 customer 信息，但不具备编辑功能*/
            <forward name="success"
            path="customer.do?method=obtainCustomers" redirect="true"/>
        </action>
    </action-mappings>
    <controller>
        <set-property property="processorClass"
        value-"org.springframework.web.struts.DelegatingRequestProcessor"/>
        <set property property-"maxFileSize" value-"2M"/>
        <set-property property-"inputForward" value "true"/>
    </controller>
```

```
    <message resources parameter "i18n/messages"/>
    <plug in className "org.springframework.web.struts.
    ContextLoaderPlugIn"/>
    <plug-in className-"org.apache.struts.validator.ValidatorPlugIn">
        <set-property property-"pathnames"
                     value="/WEB-INF/validator-rules.xml,/WEB-INF/
                     validation.xml"/>
    </plug-in>
</struts-config>
```

现在就可以在客户端显示为结果为如图 8-12 和图 8-13 所示，点击图 8-12 中的“detail”可得图 8-13 所示的界面。

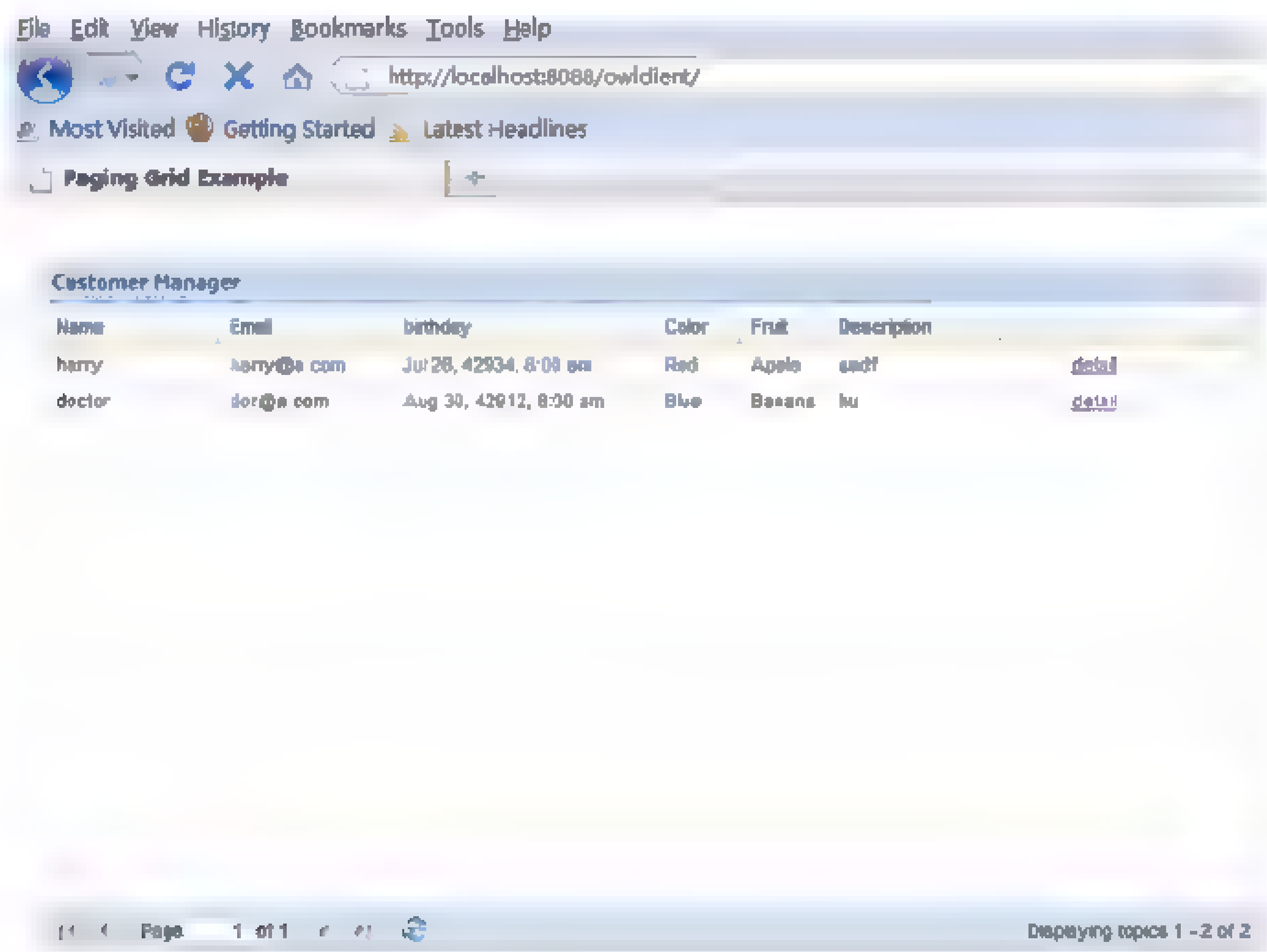

图 8-12　应用实例客户端显示结果

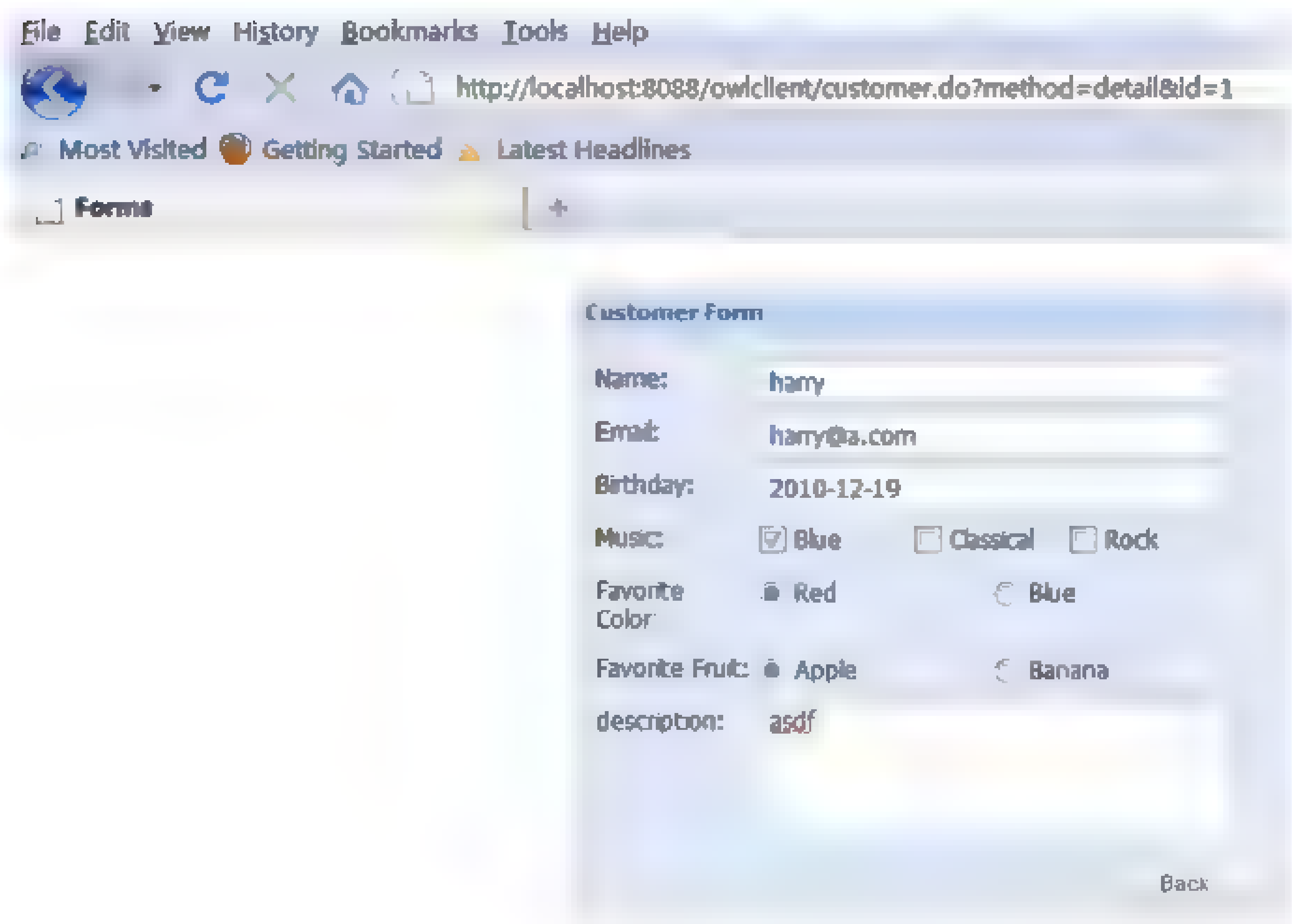

图 8-13　应用实例在客户端显示结果

小　　结

本章将 SAJP-M 中间件框架进行了详细的分析，并且采用一个客户信息和偏好的应用实例进行了详尽的描述。同时，对涉及到的重要代码也进行分析，特别地，此中件间不旦整合了 SSH，还整合了 A2JT，并且通过这些整合的详细分析，很容易实现基于 Spring 的其他开源软件整合（如图 8-1 所示），以及其他开源软件框架的整合。当然在本书前面相关章节也分析了其他开源软件与 Spring 的整合方法。其中 A2JT 的整合是近来相关书籍、资料介绍较小的，而本书针对此，不旦进行整合了，还通过一个非常有用且参考价值的应用实例进行详细的分析和讲解。在这个例子应用了界面开源软件 EXT，但 EXT 无法直接访问 bean，这时需要通过 JSON 进行读取。因此 A2JT 不仅整合了 CXF、Spring 和 WSDL2OWL-S，还整合了漂亮且实用的界面开源软件 EXT，这对研发面向服务的美观系统提高了有效的参考。同时，在应用实例中，全面应用了 SAJP-M 中间件，这为第 9 章基于该中间开发一个科研绩效系统提供铺垫。

第9章　用SAJP-M设计实现科研绩效系统

本章就基于SAJP-M中间件研发一个科研绩效管理系统，该系统主要是基于SSH来实现，并借助存储过程实现万级单位的科研数据统计分析。同时，该系统也是对本书所涉及的相关内容进一次系统的总结，从而体现面向Web开源软件的整合研发实际应用软件系统的模式。

9.1　系统描述

基于J2EE构建的开源软件（OSS，Open Source Software）技术已在各行各业的管理信息化系统、资源管理系统、情报信息处理、知识管理等中得到了充分的利用，这是由于OSS具有安全性、可靠性、稳定性、开放标准、无知识产权纠纷、软件本地化、不依赖软件供应商等优点。并且在学术界进行了大量的研究，如面向技术接受模型个体属性的变化和因素鉴定的用户接受模型；为改进开发者和用户使用OSS的效率。因此，利用OSS实现各行各业信息化系统建设将具有独特的优势，而本章就是利用OSS来设计实现一款科研绩效管理系统，而科研管理方法和策略一直都是应用界和学术界研究的方向[1]；科研管理信息建设是提高科研管理的科学化管理水平[2]，因此相关的管理信息化系统在各单位都已实现了应用，所采用的技术一般有J2EE、.NET和PHP技术等。但由于科研管理中的统计分析是一项繁杂的工作，这是因为不同的单位对科研统计分析的需求是不一样的，而且多数科研成果是由多位研究者共同完成，这就涉及到同一个成果不同的量化分配、奖励设置、集体排名、个体排名、各类成果级别等级和比例分配等问题。本章通过利用OSS分析设计出一款科研统计分析系统来（以高校科研统计分例）解决这些问题，优点在于：操作方便、数据处理速度快、“一键式统计分析”（即只需要科研管理人员操作一个按钮就可以进行统计分析）、计算错误检测等。

同时，科研绩效管理的统计分析中涉及计算用存储过程代替业务逻辑计算。这样就能提高科研数据统计分析效率，简化编程流程。而存储过程是SQL语句的集合，具有在存储过程中声明和设置SQL变量、实现流程控制、处理异常，能够对数据进行插入、更新、删除，以及传递和返回结果集的功能。因此具备较高的运行效率、减少服务器和客户之间的信息交换、确保数据访问安全和提高数据共享等优点。而存储过程还有具有SQL透明访问远程数据源上数据，且返回的数据还包含输出参数、具体返回值和结果集，这大大丰富的存储过程的使用范围，使存储过程更具有可移植性、可重用性、安全性和可伸缩性等独特的优势。因此在数据处理相关等领域得到了广泛和成功的使用，主要表现在数据访问、数据抽取、数据清洗、数据存储、数据交换等[1]。而本章就采用存储过程的来实现科研绩效管理的统计分析设计，这是因为科研统计涉及的计算量非常大、计算规则复杂。在过去很多科研统计分析系统一直都采用业务逻辑来实现科研统计分析功能和规则约束功能[3,4]，导致了计算速度慢，Java类的程序冗余性强，输出结果难满足科研管理的需求。而本章的科研统计分析、计算机规则和报表输出功能都采用存储过程来实现，使用整个科研管理系统的统计分析集中一起，从而提高

科研成果统计的速度和准确性，降低业务逻辑的复杂性。

9.2　系统需求

根据当前科研统计分析的状况可以分析出基本需求包括：基于 Web 的分布式科研信息录入与处理，友好的操作界面，动态科研统计分析的参数设置，操作简单方便，科研统计功能完备，统计分析的计算速度快且准确，最终报表输出功能齐全，安全的开发技术。因此：

（1）结合 AJAX、CSS 技术对操作页面进行设计，提高系统操作的方便性，并且增强页面的美感度；进而也提高了页面异步处理能力。

（2）整个系统采用模块化设计，并用开源软件技术实现系统开发，有效增加系统的安全性。

（3）按科研统计的类别、性质、特征、各种参数等，动态设置各类参数的设置方法。

（4）根据需求设置各种科研统计分析的规则，用优化存储过程编写统计分析计算方法，并按部门和个人两类进行排名打印输出，这样能有效提高多数据的处理速度。

（5）制定科研统计分析的各种规范和规则，提高统计分析的精确性。

（6）将统计分析功能集中在一个按钮上去执行基于存储过程科研统计分析方法，并设置检测错误的功能，这样是降低计算的复杂性和计算的正确性。

9.3　系统构架

系统功能主要由科研基本信息初始化模块、科研信息录入模块、科研成果录入模块、系统参数设置模块和统计分析模块五大模块组成，如图 9-1 所示：

9.3.1　数据库构架

科研绩效管理系统共 23 个基本数据表组成，这些表包括它们分别是科研成果表（LHHD_PRT）、登录信息记录表（MSG_LOG）、系统菜单（T_MENU）、奖励表（T_BONUS）、会议类型表（T_CNFERENCE_TYPE）、部门信息表（T_DEPT_INFO）、员工信息表（T_FACULTY）、专著类型表（T_MONOGRAPH_TYPE）、文章检索信息表（T_PAPER_INDEX）、期刊信息表（T_PERI_INFO）、期刊类型表（T_PERI_TYPE）、职称表（T_PROFESSIONAL）、科研比例类型表（T_PROFESSION）、比例信息表（T_PROFESSION_RULE）、科研成果记录表（T_RECORD）、科研成果作者关系表（T_RECORD_AUTHOR）、科研成果与单位记录表（T_RECORD_FIRM）、记录与科研成果、检索关系表（T_RECORD_PAPER_INDEX）、科研奖励类型表（T_RESULT_TYPE）、学校表（T_SCHOOL）、学科分类表（T_SUBCLASS）等，具体说明见表 9-1 所示，如创建 T_MENU 的格式（如下程序代码），相应其他表的创建也是按此种格式编写的，详见赠送光盘。这些表分别用来存储和记录科研统计分析所需求的基础数据和参数数据。主要包括成果录入的载体数据、成果存储数据、基本信息基础数据、各类系统参数设置数据和报表输出数据等，且这些表的逻辑结构如图 9-2 所示，图 9-3 所示是数

据库安装成功的界面。

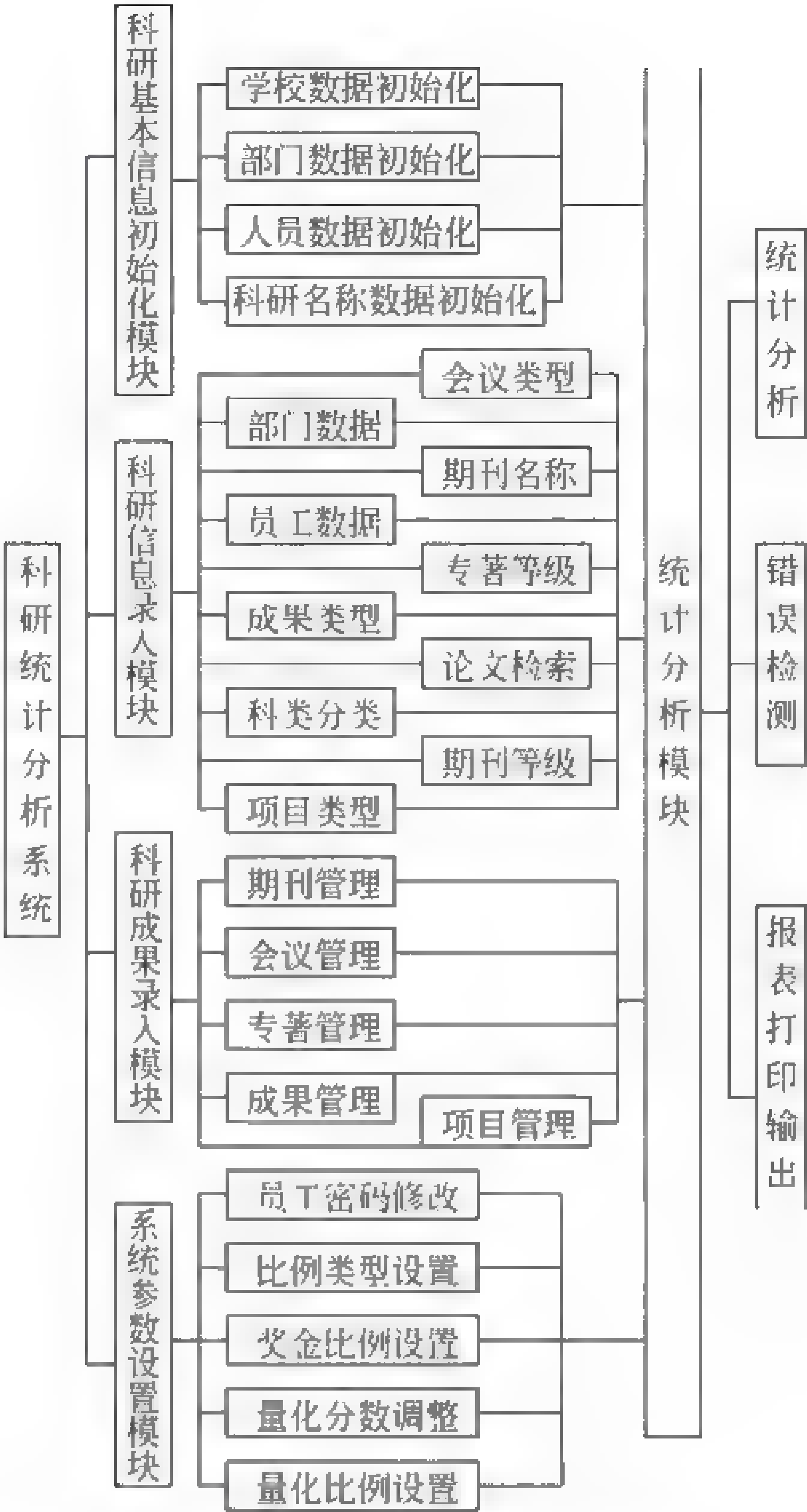

图 9-1　科研绩效管理系统功能组成

表 9-1　科研绩效统计分析表

表名	基本描述
LHHD_PRT	科研成果表，用于存储科研成果
MSG_LOG	登录信息记录表，用于记录登录信息
T_MENU	用于动态存储系统菜单
T_BONUS	奖励表，用于存储奖励信息
T_CNFERENCE_TYPE	用于科研成果之会议类型
T_DEPT_INFO	部门信息表，用于存储部门信息
T_FACULTY	员工信息表
T_MONOGRAPH_TYPE	用于科研成果之专著类型表
T_PAPER_INDEX	文章检索信息表
T_PERI_INFO	用于科研成果之期刊信息表
T_PERI_TYPE	期刊类型表
T_PROFESSIONAL	员工职称信息表
T_PROFESSION	科研成果分配比例类型表
T_PROFESSION_RULE	科研成果比例信息表

续表

表名	基本描述
T_RECORD	科研成果记录表
T_RECORD_AUTHOR	科研成果作者关系表，用于建立科研与作者的联系
T_RECORD_FIRM	科研成果与单位记录表
T_RECORD_PAPER_INDEX	记录与科研成果、检索关系表
T_RESULT_TYPE	科研奖励类型表
T_SCHOOL	不同单位表
T_SUBCLASS	学科分类表

T_MENU 数据库创建代码如下：

```
CREATE TABLE ABSZ.T MENU(
  MENU ID NUMBER(19,0) NOT NULL ENABLE,
    ADMIN PRIV FLAG NUMBER(1,0),
    FACULTY_PRIV_FLAG NUMBER(*,0) DEFAULT 0,
    LEADER PRIV FLAG NUMBER(1,0),
    MENU NAME VARCHAR2(255 CHAR),
    MENU URL VARCHAR2(255 CHAR),
    MENU PARENT ID NUMBER(19,0),
     PRIMARY KEY (MENU ID)
  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  TABLESPACE ABSZ_INDEX  ENABLE,
     CONSTRAINT FKCB5FF64A52F12829 FOREIGN KEY (MENU PARENT ID)
      REFERENCES ABSZ.T MENU (MENU ID) ENABLE
   ) PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 NOCOMPRESS LOGGING TABLESPACE
   ABSZ_DATA;
```

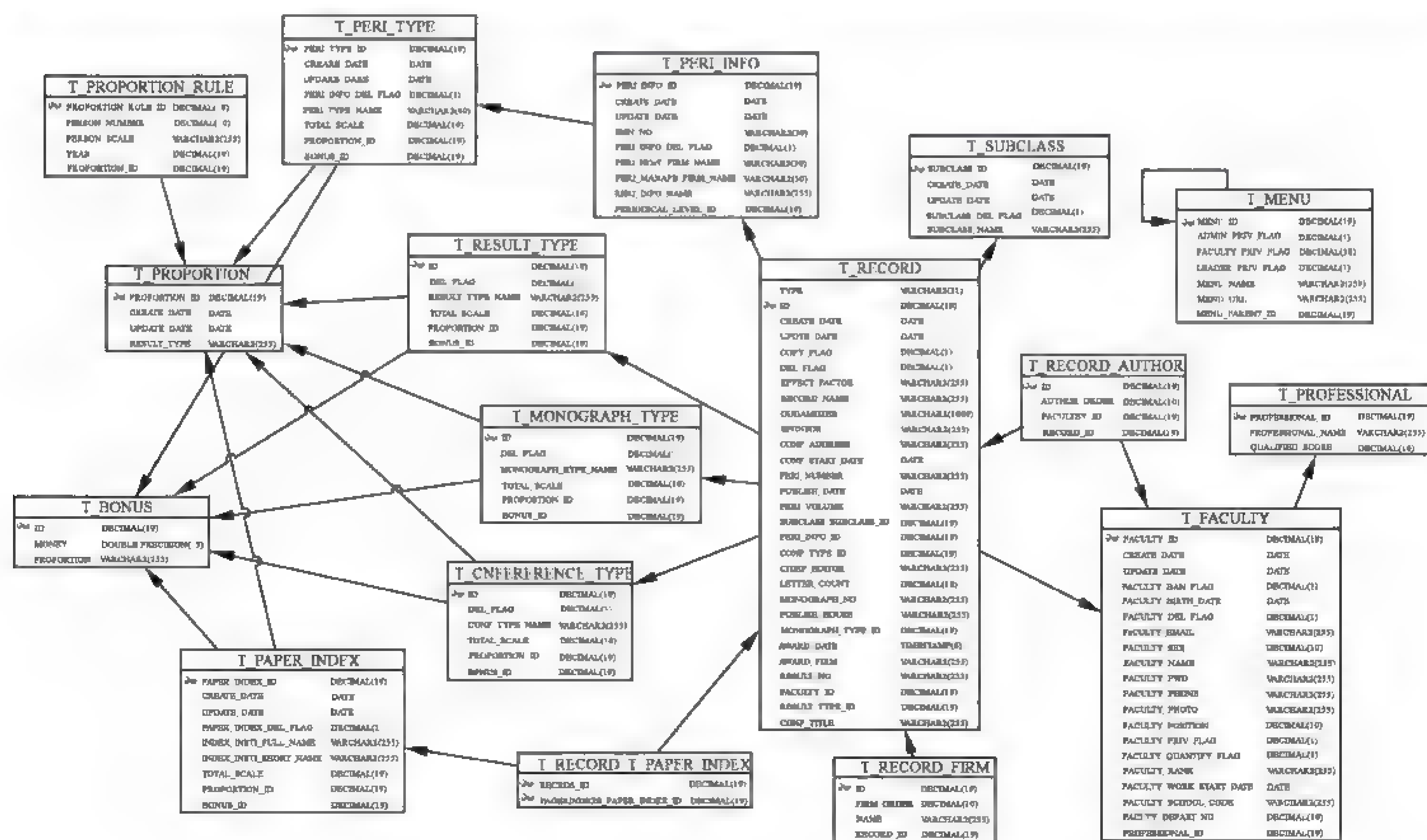

图 9-2　科研绩效管理统计分析系统数据库逻辑关系图

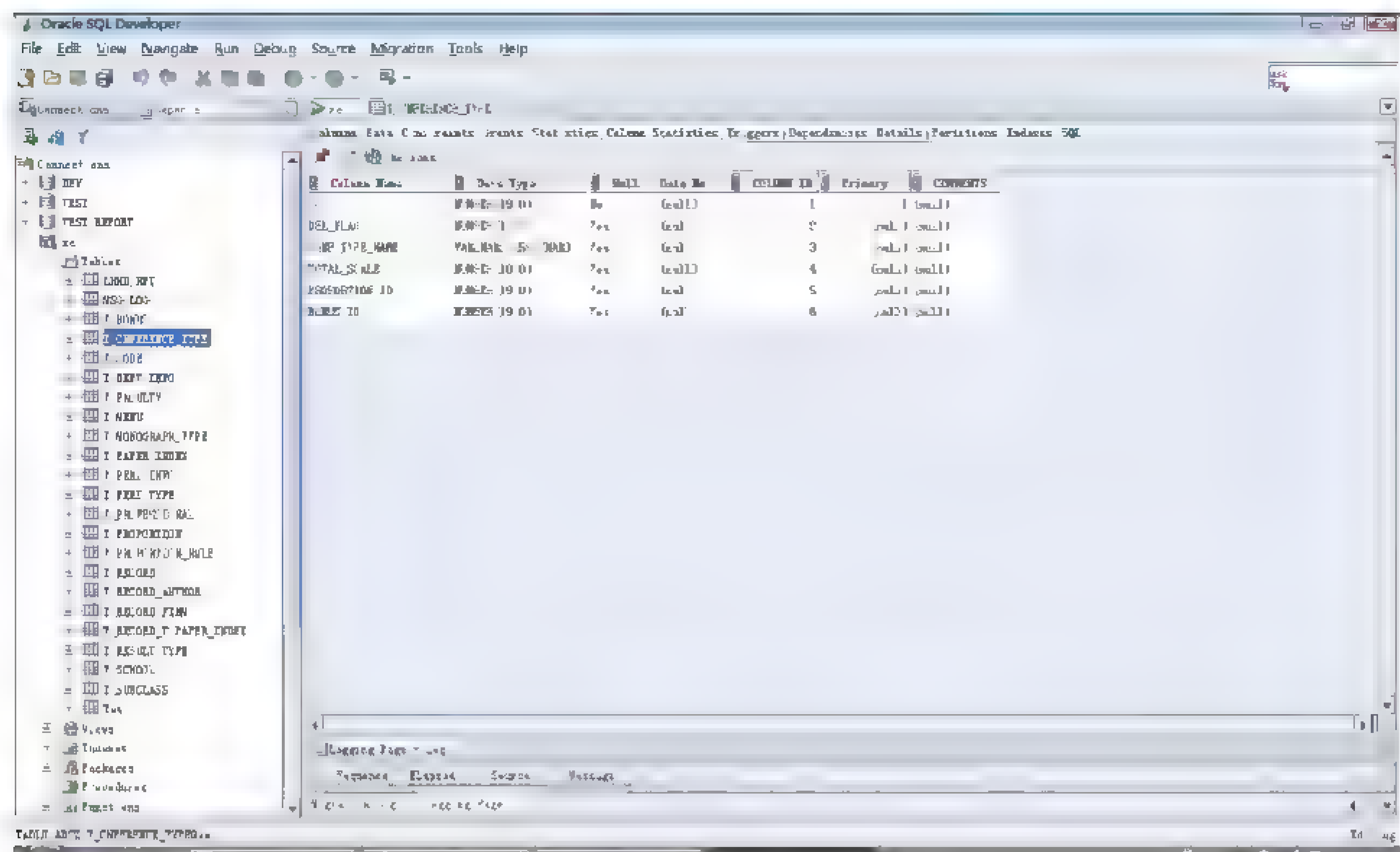

图 9-3　数据库安装成功界面（OracleXE 和 Oracle sqldeveloper 为例）

9.3.2　系统构架

根据的系统的需求，设置三个角色，分别是科研管理员角色、院系管理员角色，教职工角色。并根据这些角色的操作权限动态分配各自的功能，如图 9-4 所示：

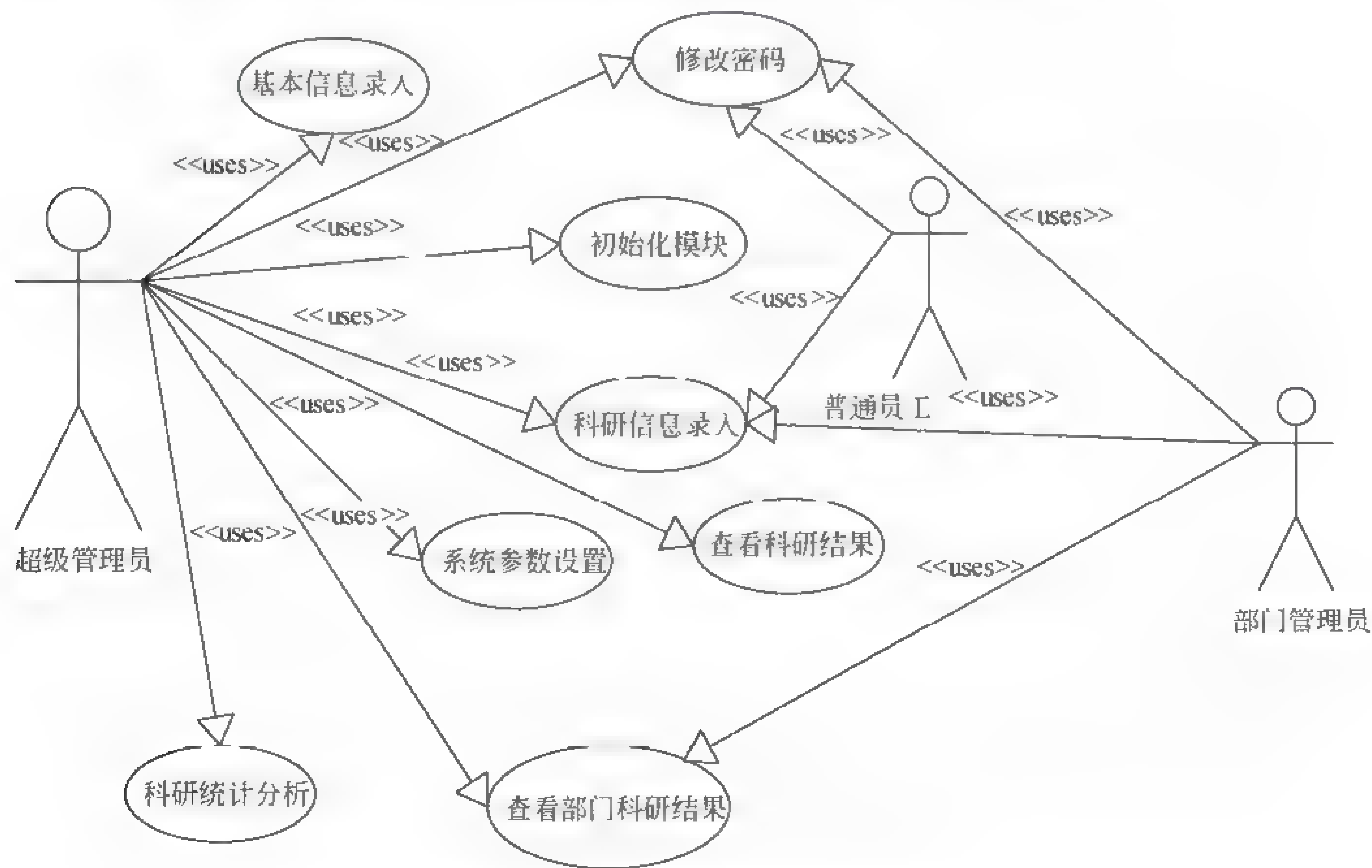

图 9-4　不同角色操作系统的用例图

而科研绩效管理功能模块的用例图如图 9-5 所示，它描述整个系统的功能关系，以及这些功能在整个系统中的位置。

图 9-5　科研绩效统计分析功能用例图

一般情况下，很多科研绩效统计分析系统的功能都是以业务逻辑的形式来实现的，这虽然降低了系统实现的难度，但是影响了统计分析的速度。因此，本章的科研绩效管理的统计分析采用存储过程的思想来实现。

系统通常统计分析的数据包括期刊论文、专著教材、科研奖励、会议论文、各类政府性质的竞赛奖励、项目申请结题。针对论文有期刊论文的特征有所属期刊级别、被检索机构检索类型、作者排名次序、作者单位次序，专著教材的特征有出版社级别、字数、作者排名次序、作者单位次序，科研奖励的特征有奖励级别、等级、作者排名次序、作者单位次序，会议的特征有会议级别、被检索机构检索类型、作者排名次序、作者单位次序，各类政府性质的竞赛奖励的特征有获奖级别、等级、指导教师排名次序，项目申请结题特征有级别、资助金额、完成状态、产出成果质量和数量。这时就要根据这些成果的特征对每一项成果的量化绩效分数、奖金额度按所设比例进行分配，并将分配的结果按部门与个人排名输出，按部门输出结果如图 9-6 所示，其科研统计分析输出存储过程代码详见附赠光盘，其通过一个名为 absz lhhd pkg bd.sql 的存储过程来实现统计分析，该存储过程代码 1000 多行，主要从同一科研成果多人分配中所涉及到成果等级、论文检索级别、作者排名次序、奖金分配情况、量化计算分配、单位次序角度来进行全方位的计算，并最后按个人和部门从高至低进行排名进

行输出。

图 9-6　系统以部门统计分析结果界面

此时分配规划约束如下：

（1）针对量化分数规则（以下规则都按作者次序和单位次序分配）：

① 若期刊论文具有检索情况，也有期刊等级情况，就按量化分数高的计算。

② 若科研奖励具有级别情况，也有等级情况，就按级别和等级所属量化分数高的计算。

③ 若成果是专著教材，则按出版社等级高的分数计算。

④ 若成果是会议论文；若会议论文未被检索机构检索，则按会议级别高的分数计算；若会议论文被检索机构检索，则按检索机制级别高的分数计算。

⑤ 若科研项目，则按项目级别或资助金额的计算。

（2）针对奖金额度规则：

① 如果第一作者是本校老师，就全部给第一作者，不够考虑后面的。

② 第一作者不是本校老师，若第二作者是本校老师，第三作者又不是本校老师，则按一定的比例把奖金分给第二作者。

③ 第一作者不是本校老师，若第三作者是本校老师，第二作者又不是本校老师，则按一定的比例把奖金分给第三作者。

④ 第一作者不是本校老师，若第二、三作者都是本校老师，则把奖金按比例全分给第二作者。

⑤ 若每一项科研成果有多个作者，则只考虑前三个作者。

9.4　系统程序结构

前面三节对科研绩效管理系统的整体结构进行了描述和构架，本节就根据此分析其实现的程序结构，其程序结构如图 9-7 所示：

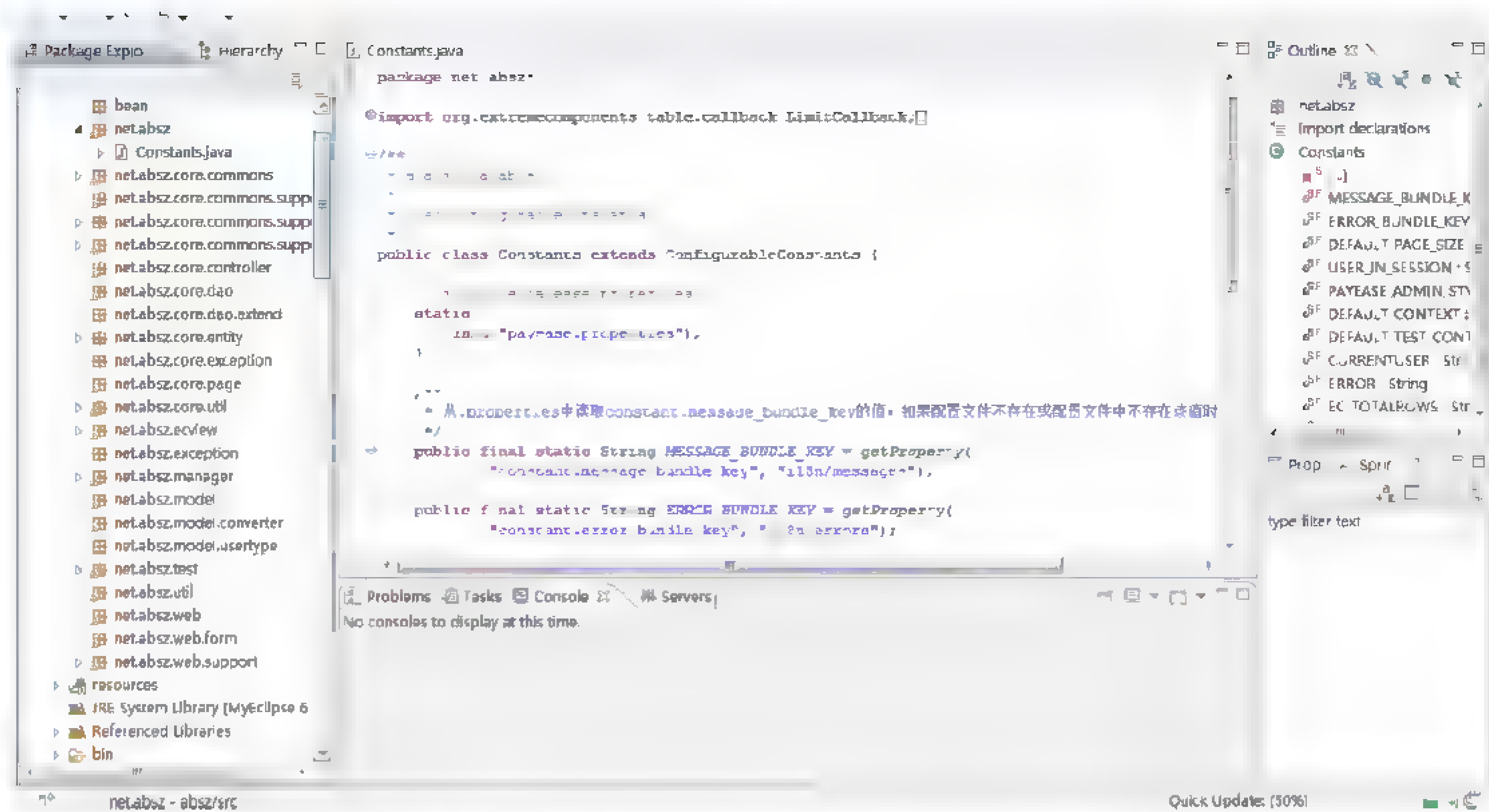

图 9-7　科研绩效系统在 IDE 环境中的程序结构

9.4.1　程序结构

系统的程序结构可用为三部分，一部分是资源与配置文件，一部分是功能实现的源代码，最后一部分是数据库与数据处理。

9.4.2　再述配置文件

前面基础知识分析了配置文件的主要是格式、结构和语法，包括在第 8 章的例子中相关的配置也是偏少的。因此，本节将继续叙述应用系统的配置文件模式，当然该模式与前面叙述是一样的，只是在结构上是有所不同的。其实整个配置文件就是科研绩效系统的核心，通过这些配置就可以大致明白该系统程序结构及实现功能情况。科研绩效统计分析系统（简称系统）的主要配置文件介绍如下：

1. action-servlet.xml

该文件配置了整个科研绩效统计分析系统的 bean，每一项的具体含义可以参见前面章节分析，不过，在系统中的 validation.xml、web.xml 文件的配置基本是一致的。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
"http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans default-autowire="byName" default-lazy-init="true" >
   <bean name="/menu" class="net.absz.web.MenuAction" scope="singleton" />
   <bean name="/faculty" class="net.absz.web.FacultyAction" scope=
     "singleton" />
```

```
        <bean name="/login" class="net.absz.web.LoginAction" scope="singleton" />
        <bean name="/article" class="net.absz.web.ArticleAction" scope="singleton" />
        <bean name="/upload" class="net.absz.web.ImportExcelAction" scope=
        "singleton" />
        <bean name="/peri" class="net.absz.web.PeriodicalAction" scope=
        "singleton" />
        <bean name="/util" class="net.absz.web.UtilAction" scope="singleton" />
        <bean name="/dept" class="net.absz.web.DepartmentAction" scope="singleton" />
        <bean name="/periInfo" class="net.absz.web.PeriInfoAction" scope=
        "singleton" />
        <bean name="/periodicalLevel" class="net.absz.web.PeriodicalLevelAction"
        scope="singleton" />
        <bean name="/paperIndexB" class="net.absz.web.PaperIndexAction" scope=
        "singleton" />
        <bean name="/subClass" class="net.absz.web.SubClassAction" scope=
        "singleton" />
        <bean name="/conferenceType" class="net.absz.web.ConferenceTypeAction"
        scope="singleton" />
        <bean name="/conference" class="net.absz.web.ConferenceAction" scope=
        "singleton" />
        <bean name="/commonProportion"
        class="net.absz.web.CommonProportionAction" scope="singleton" />
        <bean name="/proportion" class="net.absz.web.ProportionAction" scope=
        "singleton" />
        <bean name="/proportionRule" class="net.absz.web.ProportionRuleAction"
        scope="singleton" />
        <bean name="/bonus" class="net.absz.web.BonusAction" scope="singleton" />
        <bean name="/professional" class="net.absz.web.ProfessionalAction" scope=
        "singleton" />
        <bean name="/resultType" class="net.absz.web.ResultTypeAction" scope=
        "singleton" />
        <bean name="/monographType" class="net.absz.web.MonographTypeAction"
        scope="singleton" />
        <bean name="/result" class="net.absz.web.ResultAction" scope="singleton" />
        <bean name="/monograph" class="net.absz.web.MonographAction" scope=
        "singleton" />
        <bean name="/report" class="net.absz.web.ReportAction" scope="singleton" />
        <bean name="/lookError" class="net.absz.web.LookErrorAction" scope=
        "singleton" />
    </beans>
```

2. struts-config.xml

该配置文件是将整个系统的 JSP 装载到 Struts 范畴中实现基于 MVC 管理，所以这个系统比前章节介绍的文件内容要多，但模式、格式都是一样的。

```
<?xml version "1.0" encoding "UTF 8" ?>
<!DOCTYPE struts config PUBLIC
        "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
        "http://struts.apache.org/dtds/struts-config 1 2.dtd">
<struts-config>
    <form-beans>
        <form-bean name="loginForm" type="org.apache.struts.validator.
        DynaValidatorForm">
            <form-property name="username" type="java.lang.String"/>
            <form-property name="password" type="java.lang.String"/>
        </form-bean>
        <form-bean name="facultyForm" type="org.apache.struts.validator.
        LazyValidatorForm">
            <form-property name="name" type="java.lang.String"/>
            <form-property name="birthDate" type="java.util.Date"/>
            <form-property name="workStartDate" type="java.util.Date"/>
            <form-property name="phone" type="java.lang.String"/>
            <form-property name="email" type="java.lang.String"/>
            <form-property name="password" type="java.lang.String"/>
            <!--<form-property name="quantifyFlag" type="java.lang.
            boolean"/>
             form-property name="professional" type="net.absz.model.
             Professional"/>
            <form-property name="dept" type="net.absz.model.Department"/>
            <form-property name="privFlag" type="java.lang.boolean"/>
            <form-property name="banFlag" type="java.lang.boolean"/>-->
        </form-bean>
        <form-bean name="articleForm" type="org.apache.struts.validator.
        LazyValidatorForm">
            <form-property name="name" type="java.lang.String"/>
            <form-property name="copy" type="java.lang.Boolean"/>
        </form-bean>
        <form-bean name="uploadForm"  type="net.absz.web.form.UploadForm"/>
        <form-bean name="deptForm" type="org.apache.struts.validator.
        LazyValidatorForm">
            <form-property name="name" type="java.lang.String"/>
            <form-property name="depNo" type="java.lang.Long"/>
        </form-bean>
        <form-bean name="periInfoForm" type="org.apache.struts.validator.
        LazyValidatorForm">
            <form-property name="name" type="java.lang.String"/>
        </form bean>
        <form-bean name "paperIndexBForm"
            type "org.apache.struts.validator.LazyValidatorForm">
            <form property name "name" type "java.lang.String"/>
```

```
        <form property name "shortName" type "java.lang.String"/>
        <form property name "totalScale" type "java.lang.Integer"/>
    </form bean>
    <form-bean name="periodicalLevelForm"
        type="org.apache.struts.validator.LazyValidatorForm">
        <form-property name="name" type="java.lang.String"/>
        <form-property name="totalScale" type="java.lang.Integer"/>
    </form-bean>
    <form-bean name="subClassForm" type="org.apache.struts.
    validator.LazyValidatorForm">
        <form-property name="name" type="java.lang.String"/>
    </form-bean>
    <form-bean name="conferenceTypeForm"
        type="org.apache.struts.validator.LazyValidatorForm">
        <form-property name="name" type="java.lang.String"/>
        <form-property name="totalScale" type="java.lang.Integer"/>
    </form-bean>
    <form-bean name="conferenceForm" type="org.apache.struts.validator.
    LazyValidatorForm">
        <form-property name="conferenceType" type="net.absz.model.
        ConferenceType"/>
    </form-bean>

    <form-bean name="resultTypeForm" type="org.apache.struts.validator.
    LazyValidatorForm">
        <form-property name="name" type="java.lang.String"/>
        <form-property name="totalScale" type="java.lang.Integer"/>
    </form-bean>

    <form-bean name="resultForm" type="org.apache.struts.validator.
    LazyValidatorForm">
        <form-property name="resultType" type="net.absz.model.
        ResultType"/>
    </form-bean>
    <form-bean name="monographTypeForm"
        type="org.apache.struts.validator.LazyValidatorForm">
        <form-property name="name" type="java.lang.String"/>
        <form-property name="totalScale" type="java.lang.Integer"/>
    </form-bean>
    <form-bean name="monographForm"
        type="org.apache.struts.validator.LazyValidatorForm">
        <form property name="monographType" type="net.absz.model.
        MonographType"/>
    </form bean>
```

```
        <form bean name "proportionForm" type "org.apache.struts.validator.
        LazyValidatorForm">
            <form property name="type" type="java.lang.String"/>
        </form bean>
        <form-bean name="commonProportionForm"
            type="org.apache.struts.validator.LazyValidatorForm">
        </form-bean>
        <form-bean name="proportionRuleForm"
            type="org.apache.struts.validator.LazyValidatorForm">
            <form-property name="personNumber" type="java.lang.Integer"/>
            <form-property name="scale" type="java.lang.String"/>
            <form-property name="year" type="java.lang.Integer"/>
        </form-bean>
        <form-bean name="bonusForm" type="org.apache.struts.validator.
        LazyValidatorForm">
            <form-property name="money" type="java.lang.Double"/>
            <form-property name="proportion" type="java.lang.String"/>
        </form-bean>
        <form-bean name="professionalForm"
            type="org.apache.struts.validator.LazyValidatorForm">
            <form-property name="name" type="java.lang.String"/>
            <form-property name="score" type="java.lang.Integer"/>
        </form-bean>
    </form-beans>

    <action-mappings>
        <action path="/professional" name="professionalForm" scope="request"
          parameter="method" validate="false">
            <forward name="list" path="/WEB-INF/pages/professional/
            professionalList.jsp"/>
            <forward name="listPage"
                path="/WEB-INF/pages/professional/professionalList.jsp"/>
            <forward name="edit" path="/WEB-INF/pages/professional/
            professionalForm.jsp"/>
            <forward name="success"
            path="/professional.do?method=doQuery" redirect="true"/>
        </action>
        <action path="/commonProportion" name="commonProportionForm" scope=
        "request"
          parameter="method" validate="false">
          <forward name="listProportion"
          path="/WEB-INF/pages/proportionRule/chooseProportion.jsp" />
        </action>
        <action path "/menu" scope "request" parameter-"method" validate
        "false">
```

```
            <forward name="menu" path="/WEB-INF/pages/menu/menu.jsp"/>
            <forward name="login" path="/WEB-INF/loginWithIframe.jsp"/>
        </action>
        <action path="/login" name="loginForm" scope="request"
        parameter="method" validate="false">
            <forward name="login" path="/loginWithIframe.jsp"/>
            <forward name="home" path="/WEB-INF/index.jsp"/>
        </action>

        <action path="/faculty" name="facultyForm" scope="request"
            parameter="method" validate="false">
            <forward name="list" path="/WEB-INF/pages/faculty/facultyList.
            jsp"/>
            <forward name="listPage" path="/WEB-INF/pages/faculty/
            facultyList.jsp"/>
            <forward name="edit" path="/WEB-INF/pages/faculty/
            facultyForm.jsp"/>
            <forward name="chgPwd" path="/WEB-INF/pages/faculty/
            facultyChgPasswd.jsp"/>
            <forward name="success" path="/faculty.do?method=doQuery"
            redirect="true" />
            <forward name="home" path="/WEB-INF/index.jsp"/>
        </action>
        <action path="/dept" name="deptForm" scope="request"
            parameter="method" validate="false">
            <forward name="list" path="/WEB-INF/pages/dept/deptList.jsp"/>
            <forward name="listPage" path="/WEB-INF/pages/dept/
            deptList.jsp"/>
            <forward name="edit" path="/WEB-INF/pages/dept/deptForm.jsp"/>
            <forward name="success" path="/dept.do?method=doQuery" redirect=
            "true"/>
        </action>
        <action path="/bonus" name="bonusForm" scope="request"
            parameter="method" validate="false">
            <!-- <forward name="edit" path="/WEB-INF/pages/bonus/
            bonusForm.jsp"/>
            <forward name="home" path="/WEB-INF/index.jsp"/> -->
            <forward name="list" path="/WEB-INF/pages/bonus/bonusForm2.
            jsp"/>
            <forward name="listPage" path="/WEB-INF/pages/bonus/bonusForm2.
            jsp"/>
            <forward name="edit" path="/WEB-INF/pages/bonus/bonusForm2.
            jsp"/>
            <forward name="success" path="/bonus.do" redirect="true"/>
        </action>
```

```
<action path="/subClass" name="subClassForm" scope="request"
    parameter="method" validate="false">
   <forward name="list" path="/WEB-INF/pages/subClass/subClassList.
    jsp"/>
    <forward name="listPage" path="/WEB-INF/pages/subClass/
    subClassList.jsp"/>
    <forward name="createOrUpd"
    path="/WEB-INF/pages/subClass/subClassForm.jsp"/>
    <forward name="success" path="/subClass.do?method=doQuery"
    redirect="true"/>
</action>
<action path="/proportion" name="proportionForm" scope="request"
    parameter="method" validate="false">
    <forward name="list" path="/WEB-INF/pages/proportion/
    proportionList.jsp"/>
    <forward name="listPage" path="/WEB-INF/pages/proportion/
    proportionList.jsp"/>
    <forward name="createOrUpd"
    path="/WEB-INF/pages/proportion/proportionForm.jsp"/>
    <forward name="success" path="/proportion.do?method=doQuery"
    redirect="true"/>
</action>
<action path="/paperIndexB" name="paperIndexBForm" scope="request"
    parameter="method" validate="false">
    <forward name="list" path="/WEB-INF/pages/paperIndex/
    paperIndexList.jsp"/>
    <forward name="listPage" path="/WEB-INF/pages/paperIndex/
    paperIndexList.jsp"/>
    <forward name="createOrUpd"
    path="/WEB-INF/pages/paperIndex/paperIndexForm.jsp"/>
    <forward name="success" path="/paperIndexB.do?method=doQuery"
    redirect="true"/>
</action>
<action path="/conferenceType" name="conferenceTypeForm" scope=
"request"
    parameter="method" validate="false">
    <forward name="list"
        path="/WEB-INF/pages/conferenceType/conferenceTypeList.jsp"/>
    <forward name="listPage"
        path="/WEB-INF/pages/conferenceType/conferenceTypeList.jsp"/>
    <forward name="createOrUpd"
        path="/WEB-INF/pages/conferenceType/conferenceTypeForm.jsp"/>
    <forward name="success"
        path="/conferenceType.do?method=doQuery" redirect="true"/>
</action>
```

```
<action path="/resultType" name="resultTypeForm" scope="request"
    parameter="method" validate="false">
    <forward name="list" path="/WEB-INF/pages/resultType/
    resultTypeList.jsp"/>
    <forward name="listPage" path="/WEB-INF/pages/resultType/
    resultTypeList.jsp"/>
    <forward name="createOrUpd"
        path="/WEB-INF/pages/resultType/resultTypeForm.jsp"/>
    <forward name="success" path="/resultType.do?method=doQuery"
    redirect="true"/>
</action>

<action path="/monographType" name="monographTypeForm"
    scope="request" parameter="method" validate="false">
    <forward name="list"
        path="/WEB-INF/pages/monographType/monographTypeList.jsp"/>
    <forward name="listPage"
        path="/WEB-INF/pages/monographType/monographTypeList.jsp"/>
    <forward name="createOrUpd"
        path="/WEB-INF/pages/monographType/monographTypeForm.jsp"/>
    <forward name="success"
        path="/monographType.do?method=doQuery" redirect="true"/>
</action>
<action path="/conference" name="conferenceForm"
scope="request" parameter="method" validate="false">
    <forward name="list" path="/WEB-INF/pages/conference/
    conferenceList.jsp"/>
    <forward name="listPage" path="/WEB-INF/pages/conference/
    conferenceList.jsp"/>
    <forward name="edit" path="/WEB-INF/pages/conference/
    conferenceForm.jsp"/>
    <forward name="success" path="/conference.do?method=doQuery"
    redirect="true"/>
    <forward name="listFaculty" path="/WEB-INF/pages/conference/
    facultyList.jsp" />
  <forward name="saveFirm" path="/WEB-INF/pages/conference/
  saveFirmResult.jsp" />
    <forward name="articleList" path="/WEB-INF/pages/conference/
    articleList.jsp"/>
</action>
<action path="/monograph" name="monographForm"
    scope="request" parameter="method" validate="false">
    <forward name="list" path="/WEB-INF/pages/monograph/
    monographList.jsp"/>
    <forward name="listPage"
```

```
        path="/WEB-INF/pages/monograph/monographList.jsp"/>
        <forward name="edit" path="/WEB-INF/pages/monograph/
        monographForm.jsp"/>
        <forward name="success" path="/monograph.do?method=doQuery"
        redirect="true"/>
    </action>
    <action path="/article" name="articleForm" scope="request"
        parameter="method" validate="false">
        <forward name="list" path="/WEB-INF/pages/peri/articleList.
        jsp"/>
        <forward name="listPage" path="/WEB-INF/pages/peri/
        articleList.jsp"/>
        <forward name="listPeriName" path="/WEB-INF/pages/peri/
        periNameList.jsp"/>
        <forward name="articleList" path="/WEB-INF/pages/peri/
        articleList.jsp"/>
       <forward name="addArticle" path="/WEB-INF/pages/peri/ addArticle.
        jsp"/>
        <forward name="edit" path="/WEB-INF/pages/peri/addArticle.
        jsp"/>
        <forward name="success" path="/article.do?method=doQuery"
        redirect="true"/>
        <forward name="listFaculty" path="/WEB-INF/pages/peri/
        addAuthor.jsp" />
        <forward name="saveFirm" path="/WEB-INF/pages/peri/
        saveFirmResult.jsp" />
    </action>
    <action path="/periInfo" name="periInfoForm" scope="request"
        parameter="method" validate="false">
        <forward name="list" path="/WEB-INF/pages/peri/periInfoList.
        jsp"/>
        <forward name="listPage" path="/WEB-INF/pages/peri/
        periInfoList.jsp"/>
        <forward name="edit" path="/WEB-INF/pages/peri/
        periInfoForm.jsp"/>
        <forward name="success" path="/periInfo.do?method=doQuery"
        redirect="true"/>
    </action>
    <action path="/proportionRule" name="proportionRuleForm"
        scope="request" parameter="method" validate="false">
        <forward name="list"
        path="/WEB-INF/pages/proportionRule/proportionRuleList.jsp"/>
        <forward name="listPage"
        path="/WEB-INF/pages/proportionRule/proportionRuleList.jsp"/>
        <forward name="createOrUpd"
```

```
        path "/WEB INF/pages/proportionRule/proportionRuleForm.jsp"/>
        <forward name "edit"
        path="/WEB INF/pages/proportionRule/proportionRuleList.jsp"/>
        <forward name-"success"
        path-"/proportionRule.do?method=doQuery" redirect="true"/>
    </action>
    <action path="/upload"  name="uploadForm"
    scope="request"  parameter="method"  validate="false">
      <forward name="upload" path="/WEB-INF/pages/importExcel/
      upload.jsp"/>
   </action>

   <action path="/peri"  name="uploadForm"  scope="request"
         parameter="method"  validate="false">
         <forward name="upload" path="/WEB-INF/pages/importExecl/
         upload.jsp"/>
   </action>
    <action path="/paperIndex"  name="uploadForm"
         scope="request"  parameter="method"  validate="false">
         <forward name="upload" path="/WEB-INF/pages/importExecl/
         upload.jsp"/>
   </action>
   <action path="/util" scope="session"  parameter="method"  validate=
   "false">

   </action>
    <action path="/periodicalLevel" name="periodicalLevelForm"
         scope="request" parameter="method" validate="false">
        <forward name="list" path="/WEB-INF/pages/peri/
        periodicalLevelList.jsp"/>
        <forward name="listPage" path="/WEB-INF/pages/peri/
        periodicalLevelList.jsp"/>
        <forward name="edit" path="/WEB-INF/pages/peri/
        periodicalLevelForm.jsp"/>
        <forward name="success"
        path="/periodicalLevel.do?method=doQuery" redirect="true"/>
    </action>
    <action path="/result" name="resultForm"
        scope="request" parameter="method" validate="false">
        <forward name="list" path="/WEB-INF/pages/result/
        resultList.jsp"/>
        <forward name="listPage" path="/WEB-INF/pages/result/
        resultList.jsp"/>
        <forward name "edit" path "/WEB INF/pages/result/resultForm.
        jsp"/>
```

```
            <forward name="success" path="/result.do?method=doQuery"
            redirect="true"/>
        </action>
        <action path="/report" scope="request" parameter="method" validate=
        "false">
            <forward name="report" path="/WEB-INF/pages/report/report.
            jsp"/>
        </action>
        <action path="/lookError" scope="request" parameter="method"
        validate="false">
        </action>
    </action-mappings>

    <controller>
        <set-property property="processorClass"
            value="org.springframework.web.struts.DelegatingRequestProcessor"/>
        <set-property property="maxFileSize" value="2M"/>
        <set-property property="inputForward" value="true"/>
    </controller>
    <message-resources parameter="i18n/messages"/>
    <plug-in className="org.springframework.web.struts.
    ContextLoaderPlugIn"/>
    <plug-in className="org.apache.struts.validator.ValidatorPlugIn">
        <set-property property="pathnames"
                      value="/WEB-INF/validator-rules.xml,/WEB-INF/
                      validation.xml"/>
    </plug-in>
</struts-config>
```

3. applicationContext.xml

该文件用配置 spring 的 AOP、IoC 及事务管理：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:aop="http://www.springframework.org/schema/aop"
      xmlns:tx="http://www.springframework.org/schema/tx"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
      http://www.springframework.org/schema/aop
      http://www.springframework.org/schema/aop/spring-aop-2.0.xsd
      http://www.springframework.org/schema/tx
      http://www.springframework.org/schema/tx/spring-tx-2.0.xsd"
      default-autowire="byName" default-lazy-init="true">
    <!-- 属性文件读入 -->
    <bean id="propertyConfigurer"
```

```
class "org.springframework.beans.factory.config.PropertyPlaceholderConfig
urer">
        <property name="locations">
            <list>
                <value>classpath*:config/jdbc.properties</value>
            </list>
        </property>
    </bean>
    <!-- 支持 @Transactional 标记 -->
    <tx:annotation-driven/>
    <!-- 支持 @AspectJ 标记-->
    <aop:aspectj-autoproxy/>
    <!-- 以 AspectJ 方式 定义 AOP -->
    <aop:config proxy-target-class="true">
        <aop:advisor
        pointcut="execution(* net.absz.manager..*Manager.*(..))"
        advice-ref="txAdvice"/>
        <aop:advisor pointcut="execution(* net.absz.core..*Dao.*(..))"
        advice-ref="txAdvice"/>
    </aop:config>
    <!-- 基本事务定义，使用 transactionManager 作事务管理，默认 get*方法的事务为
    readonly，其余方法按默认设置。默认的设置请参考 Spring 文档事务一章。-->
    <tx:advice id="txAdvice" transaction-manager="transactionManager">
        <tx:attributes>
            <tx:method name="add*" propagation="REQUIRED"/>
            <tx:method name="del*" propagation="REQUIRED"/>
            <tx:method name="update*" propagation="REQUIRED"/>
            <tx:method name="save*" propagation="REQUIRED"/>
            <tx:method name="get*" propagation="SUPPORTS" read-only="true"/>
            <tx:method name="search*" propagation="SUPPORTS" read-
            only="true"/>
            <tx:method name="find*" read-only="true"/>
            <tx:method name="*"/>
        </tx:attributes>
    </tx:advice>
</beans>
```

4．bean.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:aop="http://www.springframework.org/schema/aop"
            xsi:schemaLocation=
               "http://www.springframework.org/schema/beans
               http://www.springframework.org/schema/beans/spring beans.xsd
```

```
                http://www.springframework.org/schema/aop
                http://www.springframework.org/schema/aop/spring aop.xsd">
        <!-- 测试使用类 分别由 方法 annotationAop/xmlAop -->
        <bean id-"helloService"
            class-"bean.HelloService"/>
        <!-- annotation aop 拦截 使用@Aspect
            @Pointcut("execution(* annotationAop(..))")
            @AfterReturning("mainMethod()")
        -->
        <bean id="xmlAop"
            class="aop.AnnotationAspectJ"/>
        <aop:aspectj-autoproxy/>
        <!-- xml aop  配置文件拦截 -->
        <bean id="XmlAspectJ"
            class="aop.XmlAspectJ"/>
        <aop:config>
             <aop:aspect ref="XmlAspectJ">
                 <aop:pointcut id="mainMethod1" expression="execution(*
                 xmlAop(..))"/>
                 <aop:after-returning pointcut-ref="mainMethod1" method=
                 "goXmlAop"/>
             </aop:aspect>
         </aop:config>
</beans>
```

5. dataAccessContext-hibernate.xml//实现 spring 与 hibernate 整合

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
"http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans default-autowire="byName" default-lazy-init="true">

    <!-- 数据源定义,使用 C3P0 连接池 -->
    <bean id="dataSource"
        class="com.mchange.v2.c3p0.ComboPooledDataSource" destroy-method="close">
        <property name="driverClass" value="${jdbc.driverClassName}"/>
        <property name="jdbcUrl" value="${jdbc.url}"/>
        <property name="user" value="${jdbc.username}"/>
        <property name="password" value="${jdbc.password}"/>
        <property name="checkoutTimeout" value="100"/>
        <property name="idleConnectionTestPeriod" value="60" />
        <property name-"maxIdleTime" value="60" />
        <property name-"maxStatements" value-"100" />
        <property name-"automaticTestTable" value-"Test" />
    </bean>
    <!--Hibernate SessionFatory-->
```

```
    <bean id "sessionFactory"
    class "org.springframework.orm.hibernate3.annotation.AnnotationSession
    FactoryBean">
        <property name="dataSource" ref="dataSource"/>
        <property name="annotatedClasses">
            <list>
                <value>net.absz.model.Menu</value>
                <value>net.absz.model.Faculty</value>
                <value>net.absz.model.Author</value>
                <value>net.absz.model.Firm</value>
                <value>net.absz.model.Department</value>
                <value>net.absz.model.Periodical</value>
                <value>net.absz.model.PeriodicalLevel</value>
                <value>net.absz.model.Proportion</value>
                <value>net.absz.model.ProportionRule</value>
                <value>net.absz.model.PaperIndex</value>
                <value>net.absz.model.SubClass</value>
                <value>net.absz.model.Professional</value>
                <value>net.absz.model.Record</value>
                <value>net.absz.model.Conference</value>
                <value>net.absz.model.ConferenceType</value>
                <value>net.absz.model.Article</value>
                <value>net.absz.model.Result</value>
                <value>net.absz.model.ResultType</value>
                <value>net.absz.model.Monograph</value>
                <value>net.absz.model.MonographType</value>
                <value>net.absz.model.School</value>
                <value>net.absz.model.Bonus</value>
                <value>net.absz.model.ErrorLog</value>
            </list>
        </property>
        <property name="hibernateProperties">
            <props>
                <!--
                <prop key="hibernate.dialect">org.hibernate.dialect.
                SQLServerDialect</prop>
                <prop key="hibernate.hbm2ddl.auto">update</prop>
                <prop key="hibernate.format sql">true</prop>
                <prop key="hibernate.show sql">true</prop>
                -->
                <prop key="hibernate.connection.provider class">
                org.hibernate.connection.C3P0ConnectionProvider</prop>
                <prop key="connection.driver class">oracle.jdbc.driver.
                OracleDriver</prop>
                <prop key="hibernate.show_sql">true</prop>
```

```
                <prop key-"hibernate.dialect">org.hibernate.dialect.
                Oracle10gDialect</prop>
                <prop key="hibernate.cache.provider class">
                org.hibernate.cache.EhCacheProvider</prop>
                <prop key-"hibernate.cache.use query cache">true</prop>
            </props>
        </property>
    </bean>

    <!--Hibernate TransactionManager-->
    <bean id="transactionManager"
        class="org.springframework.orm.hibernate3.HibernateTransactionManager">
        <property name="sessionFactory" ref="sessionFactory"/>
    </bean>
</beans>
```

6．serviceContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
"http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans default-autowire="byName" default-lazy-init="true">
    <bean id="converterRegister"
    class="net.absz.model.converter.ConverterRegisterImpl" scope="singleton" />
    <bean id="SpringUtil" class="net.absz.core.util.SpringUtil" scope=
    "singleton" />
    <bean id="menuManager" class="net.absz.manager.MenuManager" scope=
    "singleton"/>
    <bean id="facultyManager" class="net.absz.manager.FacultyManager"
    scope="singleton"/>
    <bean id="periodicalManager" class="net.absz.manager.PeriodicalManager"
    scope="singleton"/>    <bean id="deptManager" class="net.absz.
    manager.DeptManager" scope="singleton"/>
    <bean id="articleManager" class="net.absz.manager.ArticleManager"
    scope="singleton"/>
    <bean id="conferenceManager"
    class="net.absz.manager.ConferenceManager" scope="singleton"/>

    <bean id="monographManager"
    class="net.absz.manager.MonographManager" scope="singleton"/>
    <bean id="monographTypeManager"
    class="net.absz.manager.MonographTypeManager" scope-"singleton"/>
    <bean id-"resultManager" class-"net.absz.manager.ResultManager"scope-
    "singleton"/>
    <bean id "resultTypeManager"
```

```
    class="net.absz.manager.ResultTypeManager" scope="singleton"/>
    <bean id="professionalManager"
    class="net.absz.manager.ProfessionalManager" scope="singleton"/>
    <bean id="periodicalLevelManager"
    class="net.absz.manager.PeriodicalLevelManager" scope="singleton"/>
    <bean id="subClassManager" class="net.absz.manager.SubClassManager"
    scope="singleton"/>
    <bean id="paperIndexManager"
    class="net.absz.manager.PaperIndexManager" scope="singleton"/>
    <bean id="conferenceTypeManager"
    class="net.absz.manager.ConferenceTypeManager" scope="singleton"/>
    <bean id="firmManager" class="net.absz.manager.FirmManager" scope=
    "singleton"/>
    <bean id="authorManager" class="net.absz.manager.AuthorManager" scope=
    "singleton"/>
    <bean id="proportionManager"
    class="net.absz.manager.ProportionManager" scope="singleton"/>
    <bean id="proportionRuleManager"
    class="net.absz.manager.ProportionRuleManager" scope="singleton"/>
    <bean id="schoolManager" class="net.absz.manager.SchoolManager" scope=
    "singleton"/>
    <bean id="bonusManager" class="net.absz.manager.BonusManager" scope=
    "singleton"/>
    <bean id="errorLogManager" class="net.absz.manager.ErrorLogManager"
    scope="singleton"/>
    <bean id="connectionProvider" class="org.hibernate.connection.
    ConnectionProviderFactory"
    factory-method="newConnectionProvider"/>
</beans>
```

9.4.3 主要功能模块之报出输出的 Action

本节只介绍三个 Action（ResultTypeAction.java、ResultAction.java、ReportAction.java）的科研绩效统计分析输出源代码。其他源代码可以根据前一节的配置文件结构和命名空间结构从赠送光盘的源代码中详细获得。

1. ResultTypeAction.java

```
package net.absz.web;
import java.util.List;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import net.absz.core.controller.StrutsEntityAction;
import net.absz.manager.BonusManager;
import net.absz.manager.ProportionManager;
```

```
import net.absz.manager.ResultTypeManager;
import net.absz.model.Bonus;
import net.absz.model.ResultType;
import org.apache.commons.lang.StringUtils;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
public class ResultTypeAction extends
        StrutsEntityAction<ResultType, ResultTypeManager> {
    //继承 StrutsEntityAction
    private final static Log log = LogFactory.getLog(ResultTypeAction.class);
    private ResultTypeManager resultTypeManager;
    private ProportionManager proportionManager;
    private BonusManager bonusManager;
    public ActionForward createOrUpd(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {
        request.getParameter("");
        //防止重复提交的 token
        saveToken(request);
        ResultType object = null;
        //如果是修改操作，id!=null
        if (request.getParameter(idName) != null) {
            request.setAttribute("opt", "edit");
            object = doGetEntity(form, request);
            request.setAttribute("objs", object);
            if (object == null) {
                saveError(request, "entity.missing");
                return mapping.findForward(LIST);
            }
        } else {
            request.setAttribute("opt", "create");
            object = null;
        }
        List bonusLs = null;
        try {
            bonusLs = bonusManager.getAll();
        } catch (Exception e) {
            saveDirectlyError(request, "不存在可选的资金,请先添加奖金");
            return mapping.findForward(EDIT);
        }
        initForm(form, request, object);
        refrenceData(request);
```

```
        request.setAttribute("resultType", object);
        request.setAttribute("bonusLs", bonusLs);
        return mapping.findForward("createOrUpd");
    }

    public void onInitEntity(ActionForm form, HttpServletRequest request,
            ResultType object) {
        try {
            String s = request.getParameter("proportionId");
            if (s != null) {
                object.setProportion(proportionManager.findUniqueBy("id",
                        new Long(s)));
            } else {
                object.setProportion(null);
            }
        } catch (Exception e) {
            throw new RuntimeException("没有相应的比例类型存在!");
        }
        String money = request.getParameter("money");
        if (StringUtils.isNotEmpty(money)) {
            Bonus bonus = object.getBonus();
            if (null == bonus || null == bonus.getId()) {
                bonus = new Bonus();
                bonus.setMoney(Double.valueOf(money));
                getBonusManager().save(bonus);
            }
            bonus.setMoney(Double.valueOf(money));
            object.setBonus(bonus);
        }

    }
    @Override
    //声明重写方法
    public void refrenceDataC(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {
        //TODO Auto-generated method stub
        createOrUpd(mapping, form, request, response);
    }
    public ProportionManager getProportionManager() {
        return proportionManager;
    }
    public void setProportionManager(ProportionManager proportionManager) {
        this.proportionManager = proportionManager;
    }
    public BonusManager getBonusManager() {
```

```
        return bonusManager;
    }
    public void setBonusManager(BonusManager bonusManager) {
        this.bonusManager = bonusManager;
    }
    public ResultTypeManager getResultTypeManager() {
        return resultTypeManager;
    }
    public void setResultTypeManager(ResultTypeManager resultTypeManager) {
        this.resultTypeManager = resultTypeManager;
    }
}
```

2. ResultAction.java

```
package net.absz.web;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import net.absz.Constants;
import net.absz.core.controller.StrutsEntityAction;
import net.absz.manager.AuthorManager;
import net.absz.manager.FacultyManager;
import net.absz.manager.ResultManager;
import net.absz.manager.ResultTypeManager;
import net.absz.model.Author;
import net.absz.model.Department;
import net.absz.model.Faculty;
import net.absz.model.Result;
import net.absz.model.ResultType;
import net.absz.model.usertype.Position;
import org.apache.commons.beanutils.ConvertUtils;
import org.apache.commons.lang.StringUtils;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.extremecomponents.table.core.TableConstants;
import org.hibernate.Criteria;
import org.hibernate.criterion.Projections;
import org.hibernate.criterion.Restrictions;
```

```
public class ResultAction extends StrutsEntityAction<Result, ResultManager> {
    //分析统计结果
    private ResultManager resultManager;
    private ResultTypeManager resultTypeManager;
    private AuthorManager authorManager;
    private FacultyManager facultyManager;
    public FacultyManager getFacultyManager() {
        return facultyManager;
        //返回统计的教师信息
    }
    public void setFacultyManager(FacultyManager facultyManager) {
        this.facultyManager = facultyManager;
    }
    public AuthorManager getAuthorManager() {
        return authorManager;
        //返回科研成果拥有者
    }
    public void setAuthorManager(AuthorManager authorManager) {
        this.authorManager = authorManager;
    }
    public void setResultManager(ResultManager resultManager) {
        this.resultManager = resultManager;
    }
    public ResultManager getResultManager() {
        return resultManager;
    }
    public ResultTypeManager getResultTypeManager() {
        return resultTypeManager;
    }
    public void setResultTypeManager(ResultTypeManager resultTypeManager) {
        this.resultTypeManager = resultTypeManager;
    }
    @Override
    protected void afterCreatEntity(Result object, HttpServletRequest request) {
        Faculty faculty = (Faculty) request.getSession().getAttribute(
                Constants.CURRENTUSER);
        object.setFaculty(faculty);
    }

    @Override
    protected void refrenceData(HttpServletRequest request) {
        List<ResultType> resultTypes = getResultTypeManager().
        getAllForCache();
        if (null != resultTypes) {
            request.setAttribute("resultTypes", resultTypes);
```

```
        }
        String articalId = request.getParameter("id");
        if (StringUtils.isNotEmpty(articalId)) {
            Result result = getResultManager().get(Long.valueOf(articalId));
            request.setAttribute("result", result);
        }
    }

    protected ActionForward disPatchURL(String url) {
        return new ActionForward(url);
    }

    @SuppressWarnings("unchecked")
    @Override
    public ActionForward doQuery(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {
    //判断操作权限
        Faculty faculty = (Faculty) request.getSession().getAttribute(
                Constants.CURRENTUSER);
        Criteria criteria = getAuthorManager().getEntityCriteria();
        if (faculty.getPosition().equals(Position.院系领导)) {
            Department department = faculty.getDepart();
            criteria.createCriteria("faculty").add(
                    Restrictions.eq("depart", department));
        } else if (faculty.getPosition().equals(Position.教师)) {
            criteria.createCriteria("faculty").add(
                    Restrictions.eq("id", faculty.getId()));
        }
        criteria.createCriteria("record").setProjection(
                Projections.distinct(Projections.property("id")));
        List<Long> ids = criteria.list();
        if (null == ids || ids.size() == 0) {
            request.setAttribute(TableConstants.TOTAL_ROWS, new
            Integer(0));
        } else {
            Map<String, Object> filter = new HashMap<String, Object>();
            Map<String, Object[]> idIn = new HashMap<String, Object[]>();
            idIn.put("in", ids.toArray());
            filter.put("id", idIn);
            doQueryByManager(request, getResultManager(), "results", filter);
        }
        return mapping.findForward(LIST_PAGE);
    }

    protected ActionForward disPatchURL(String url, boolean redirect) {
```

```
        return new ActionForward(url, true);
    }

    public ActionForward saveArticle(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {
        String[] authorIds = request.getParameterValues("authorIds");
        String[] authorOrders = request.getParameterValues("authorOrders");
        String url = "/result.do?method=create";
        String sucessURL = "/result.do?method=doQuery";
        if (null == authorIds || authorIds.length == 0) {
            saveDirectlyError(request, "作者不能为空");
            return disPatchURL(url);
        }
        Result article;
        if (StringUtils.isNotBlank(request.getParameter(idName))) {
            article = doGetEntity(form, request);
            if (article == null) {
                saveError(request, "entity.missing");
                return disPatchURL(url);
            }
        } else {
            article = new Result();
        }
        bindBaseForm(form, article);
        Set<Author> articleAuthors = article.getAuthors();
        int length = authorIds.length;
        List<Long> existId = new ArrayList<Long>();
        for (Author  authors : articleAuthors) {
            for (int i = 0; i < length; i++) {
                if ( authors.getFaculty().getId().equals(
                        Long.valueOf(authorIds[i]))) {
                    existId.add( authors.getFaculty().getId());
                }
            }
        }
    //邦定作者与论文
        for (Author  authors : articleAuthors) {
            if (existId.size() == 0) {
                article.setAuthors(null);
                getAuthorManager().remove( authors);
            } else {
                for (Long id : existId) {
                    if (!id.equals( authors.getId())) {
                        article.setAuthors(null);
                        getAuthorManager().remove( authors);
```

```
                }
            }
        }
    }
    int authorlength = authorIds.length;
    articleAuthors = new HashSet<Author>();
    for (int i = 0; i < authorlength; i++) {
        Author author = new Author();
        Faculty faculty = getFacultyManager().get(
                Long.valueOf(authorIds[i]));
        author.setFaculty(faculty);
        author.setAuthorOrder(Integer.valueOf(authorOrders[i]));
        author.setRecord(article);
        articleAuthors.add(author);
    }
    article.setAuthors(articleAuthors);
    try {
        getEntityManager().save(article);
    } catch (Exception e) {
        saveDirectlyError(request, "保存失败,请重新保存");
        return disPatchURL(url);
    }
    return disPatchURL(sucessURL, true);
}

public ActionForward disPlayAuthorTable(ActionMapping mapping,
        ActionForm form, HttpServletRequest request,
        HttpServletResponse response) {
  //统计科研成果作者人数情况
    String[] facutyIds = request.getParameterValues("authorIds");
    String[] authorOrders = request.getParameterValues("authorOrders");
    if (null != authorOrders && authorOrders.length > 0
            && null != facutyIds && facutyIds.length > 0) {
        List<Author> authors = new ArrayList<Author>();
        int length = facutyIds.length;
        for (int i = 0; i < length; i++) {
            Author author = new Author();
            Faculty faculty = getFacultyManager().get(
                    (Long) ConvertUtils.convert(facutyIds[i], Long.class));
            author.setFaculty(faculty);
            author.setAuthorOrder((Integer) ConvertUtils.convert(
                    authorOrders[i], Integer.class));
            authors.add(author);
        }
        request.setAttribute("authors", authors);
```

```
        } else {
            Result article = (Result) request.getAttribute("result");
            if (null == article) {
                return null;
            }
            Set<Author> authors = article.getAuthors();
            request.setAttribute("authors", authors);
        }
        return new ActionForward(
                "/WEB-INF/pages/conference/displayAuthorTable.jsp");
    }
}
```

3. ReportAction.java

```
package net.absz.web;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Types;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import net.absz.Constants;
import net.absz.core.controller.StrutsAction;
import net.absz.manager.ErrorLogManager;
import net.absz.model.ErrorLog;
import net.absz.model.Faculty;
import org.apache.commons.lang.StringUtils;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.hibernate.connection.ConnectionProvider;
import org.springframework.orm.hibernate3.support.HibernateDaoSupport;

public class ReportAction extends StrutsAction {
    //将分析统计结果以报表形式输出
    private final static Log log = LogFactory.getLog(ReportAction.class);
```

```
    private final static String REPORT RESULT = "report";
    private static Map<String, Object> verifyMap;
    private static ConnectionProvider connectionProvider;
    private ErrorLogManager errorLogManager;
    public ErrorLogManager getErrorLogManager() {
        return errorLogManager;
    }

    public void setErrorLogManager(ErrorLogManager errorLogManager) {
        this.errorLogManager = errorLogManager;
    }

    public void setConnectionProvider(ConnectionProvider pConnectionProvider) {
        connectionProvider = pConnectionProvider;
    }

    public ActionForward deptReport(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {
        request.setAttribute("connectionProvider", connectionProvider);
        return new ActionForward("/WEB-INF/pages/report/deptReport.jsp");
    }
    //通过报表形式输出科研成果统计时的部门名单
    public ActionForward facultyReport(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {
        request.setAttribute("connectionProvider", connectionProvider);
        return new ActionForward("/WEB-INF/pages/report/facultyReport.jsp");
    }
    //通过报表形式输出科研成果统计时的教师名单

    private void returnVerifyResult(Map<String, Object> verifyMap,
            HttpServletResponse response) {
        PrintWriter out;
        try {
            out = response.getWriter();
            String msg = (String) verifyMap.get("info");
            out.println(msg);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    private void generateMsg(Map<String, Object> verifyMap,
            HttpServletRequest request, HttpServletResponse response) {
        Faculty user = (Faculty) verifyMap.get("user");
        Faculty currentUser = (Faculty) request.getSession().getAttribute(
```

```
            Constants.CURRENTUSER);
    String msg = null;
    if (user.getId().equals(currentUser.getId())) {
        msg = "量化核定正在被你上一次操作执行中,请等待核定完成后,才能再次执行!";
        msg = "{'status':false,'msg':'" + msg + "'}";
    } else {
        String name = user.getName();
        String id = user.getId().toString();
        msg = "量化核定正在被[教职工:" + name + "(教职工ID:" + id
                + ")] 执行,请等待对方核定完成后,再执行量化核定!";
        msg = "{'status':false,'msg':'" + msg + "'}";
    }
    verifyMap.put("info", msg);
    returnVerifyResult(verifyMap, response);
}

public void checkVerify(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response) {
    response.setCharacterEncoding("UTF-8");
    String successJson = "{'status':true}";
    if (null == verifyMap) {
        synchronized (this.clazz) {
            if (null == verifyMap) {
                verifyMap = new HashMap<String, Object>();
                Faculty user = (Faculty) request.getSession().getAttribute(
                        Constants.CURRENTUSER);
                verifyMap.put("user", user);
                verifyMap.put("info", successJson);
                returnVerifyResult(verifyMap, response);
                return;
            }
        }
    }
    generateMsg(verifyMap, request, response);
}

public void doVerify(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response) {
    verifyMap = null;
    response.setCharacterEncoding("UTF-8");
    String successJson = "{'status':true}";
    String errorJson = "{'status':false,'msg':'基础数据不全,请到错误信息菜
    单查看有问题的记录!'}";
    PrintWriter out;
    try {
```

```
            out = response.getWriter();
            if (verify()) {
                out.println(successJson);
            } else {
                out.println(errorJson);
            }
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    private boolean verify() {
        Connection con = null;
        String valid = "";
        try {
            con = connectionProvider.getConnection();
            String st = "{call absz lhhd.prc lhhd(?,?)}";
            CallableStatement callsta = con.prepareCall(st);
            callsta.setInt(1, 2009);
            callsta.registerOutParameter(2, Types.CHAR);
            callsta.execute();
            valid = callsta.getString(2).trim();
        } catch (SQLException e) {
            throw new RuntimeException(e);
        } finally {
            try {
                connectionProvider.closeConnection(con);
            } catch (SQLException e) {
                throw new RuntimeException(e);
            }
        }
    //验证科研统计分析日期
        if (valid.equals("1")) {
            return true;
        }
        return false;
        //测试调用执行成功与否
        //System.out.println(callsta.execute());
        //循环输出调用存储过程的记录结果
        /**
         * ResultSet re; if (callsta.execute() == true) { re =
         * callsta.getResultSet(); while (re.next()) {
         * System.out.println(re.getInt(1) + " " + re.getString(2)); } }
         */
    }
```

```
    public void hasVerified(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response)
            throws IOException {
        response.setCharacterEncoding("UTF-8");
        String year = request.getParameter("year");
        String successJson = "{'status':true}";
        String errorJson = "{'status':false,'msg':'参数不正确!'}";
        PrintWriter out = response.getWriter();
        if (StringUtils.isEmpty(year)) {
            out.write(errorJson);
            return;
        }
        List<ErrorLog> years = getErrorLogManager().findBy("year", year);
        if (null != years && !years.isEmpty()) {
            out.write(successJson);
        } else {
            String error = "{'status':false,'msg':'此年份未核定,请先核定后再打
            印!'}";
            out.write(error);
        }
    }

    public ActionForward chooseReport(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {
        return new ActionForward("/WEB-INF/pages/report/chooseReport.jsp");
    }
  //输出统计报表
    public ActionForward unspecified(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {
        return new ActionForward("/WEB-INF/pages/report/verify.jsp");
    }
  //验证输出统计报表
    public ActionForward doProcessing(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {
        return new ActionForward("/WEB-INF/pages/report/processing.jsp");
    }
    class ReportHibernate extends HibernateDaoSupport {
        Connection con;
        @SuppressWarnings("deprecation")
        ReportHibernate() {
        }
        void getReport(String sql) throws SQLException {
            Statement st = con.createStatement();
            ResultSet rs = st.executeQuery(sql);
        }
```

```
        }
    }
```

9.4.4　系统运行

要运行系统，首先需要安装数据库。本系统的数据库安装是通过 bat 来实现的，其结果如图 9-8 所示。

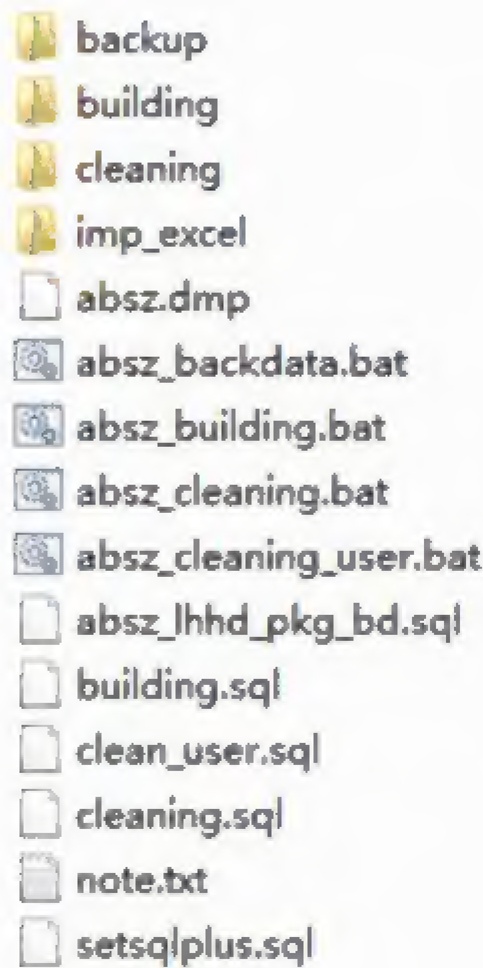

图 9-8　数据库结构

（1）双击图 9-8 中的 absz_building.bat 文件，它将调用 building.sql 执行一系列脚本建立用户名和密码以及表空间和初始化数据，初始化用户名和密码为 absz。该用户是整个系应用访问后台数据库的帐号（即超级管理员）。若顺利完成运行，就表明数据库已成功建成。

（2）检查各配置文件，特别是 web.xml 和 server.xml 文件配置是否正确。

（3）启动 Web 服务器，此处采用 Tomcat，相关 Tomcat 的原理在前面章节已有分析。此时：

① 找开登录界面如图 9-9 所示。

图 9-9　科研绩效管理统计分析系统登录页面

② 输入超级管理员名和密码，得到图 9-10 所示的页面。

图 9-10　科研绩效管理统计分析系统界面（示例）

③ 如图 9-11 所示是添加科研成果之论文的界面。

图 9-11　科研绩效管理统计分析系统的添加科研成果之论文操作页面（示例）

④ 图 9-12 是科研成果比例分配页面。

⑤ 图 9-13 是科研绩效统计分析页面。

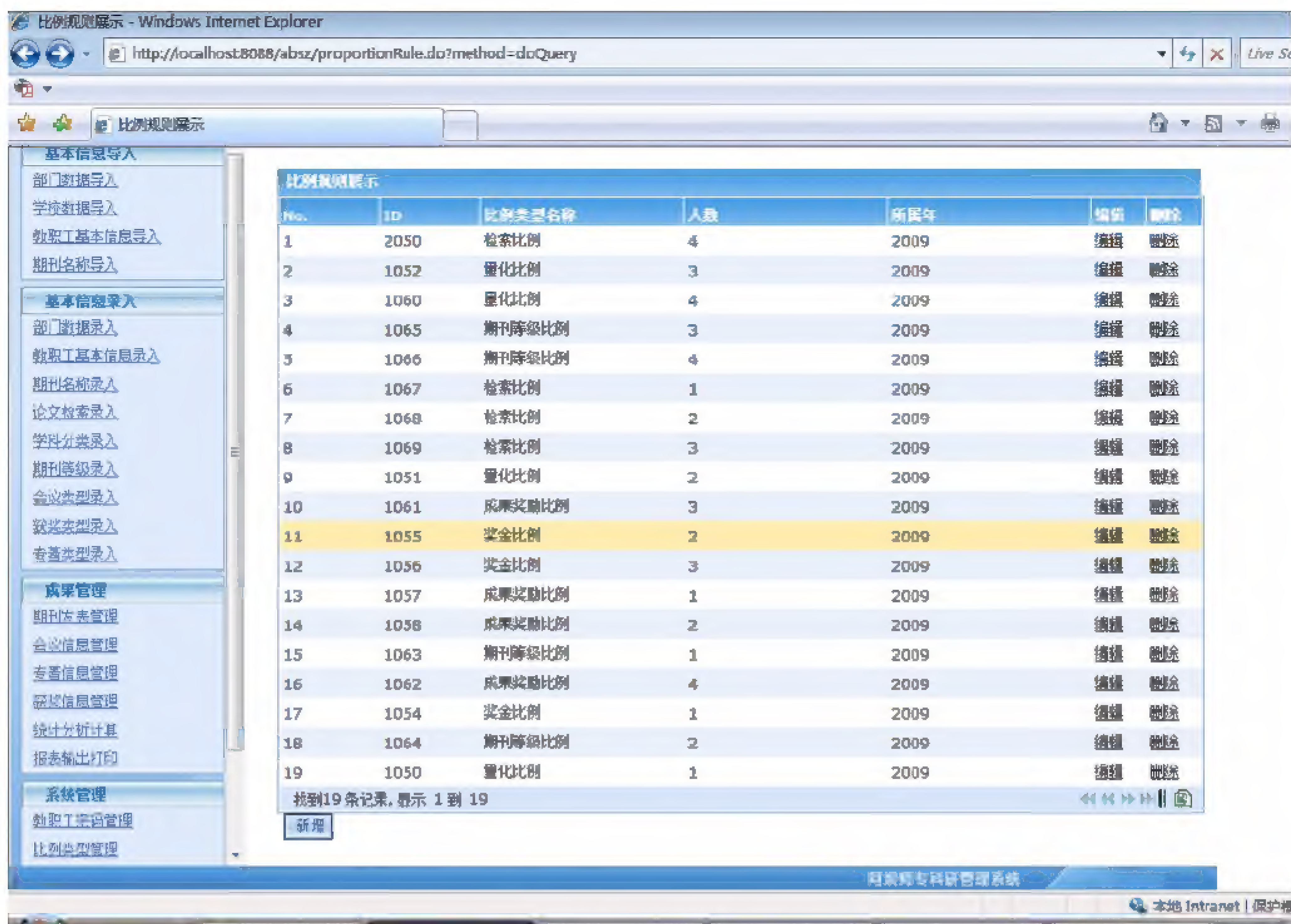

图 9-12　科研绩效统计分析系统的科研成果比例分配页面

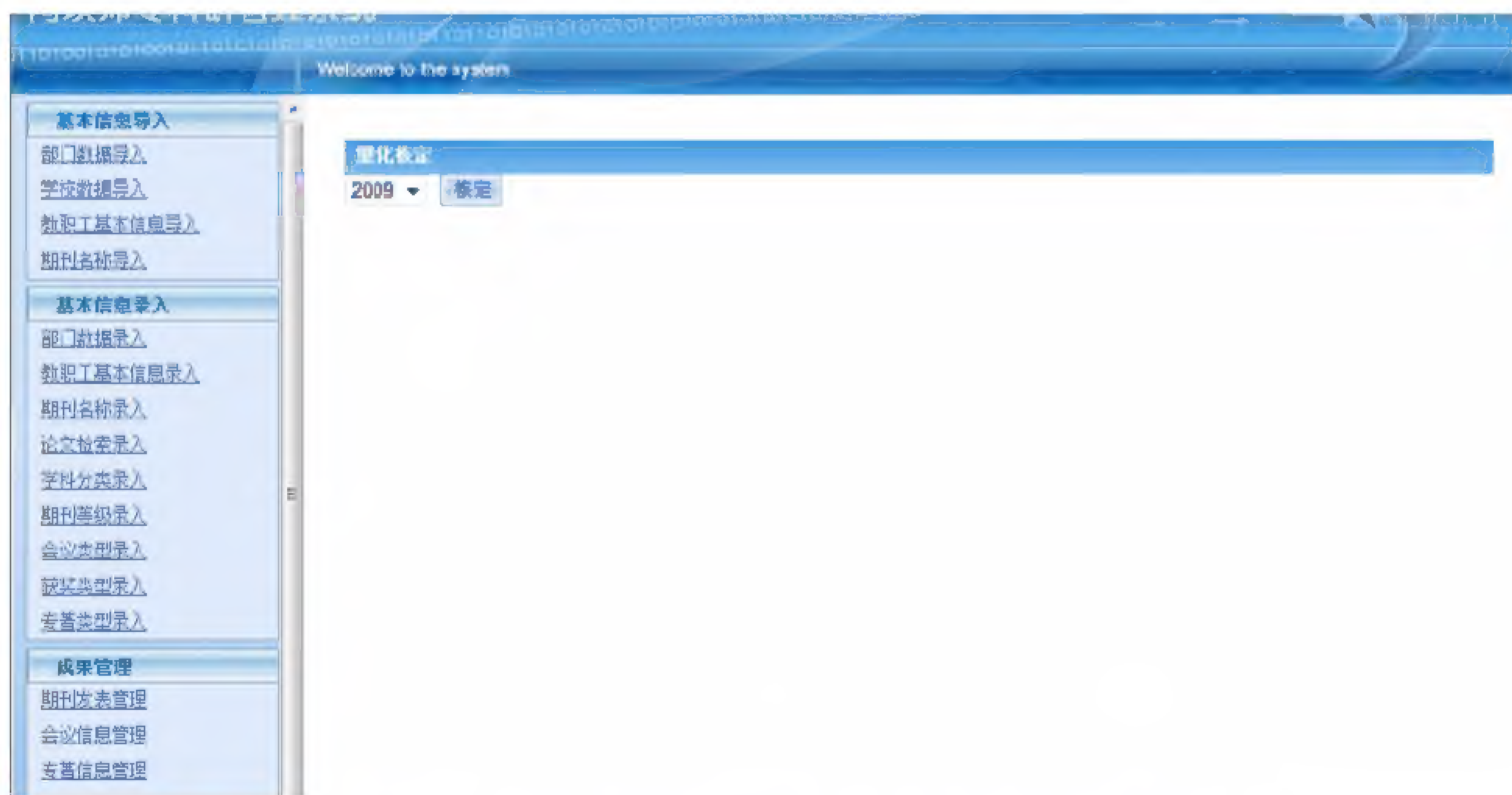

图 9-13　科研绩效统计分析系统的科研成果分析页面

小　　结

本章叙述了基于 SAJP-M 的科研绩效管理分析统计系统，即该系统已研发实现。即主体是采用开源软件 SSH 整合实现，这也是比第 8 章实用实例更为完整的一个系统描述。在第 8

章中的侧重点主要倾向于A2JT的开发举例，在第9章中，则侧重于SSH的应用研发。因此，这两个例子有效的体现本书的整合与实战应用。

在本章主要分析了该系统的需要分析、功能分析设计，数据库分析设计等，也再一次分析描述了开源软件整合与应用的配置文件。使其配置文件不再停留在基本模式而和结构上，而体现具体的应用系统应用上。因此，再一次详细的列举了配置文件，以及列举了科研统计分析的报表输出的Action代码。这样可以使读者更易于明白整个的结构，以及功能模块与程序结构。

参 考 文 献

[1] 孙佰清. 高等学校科研管理绩效定量评价方法研究. 哈尔滨工程大学学报，2010，31(6)：803-808.

[2] 马建霞，祝忠明，唐润寰，等. 机构知识库与科研管理信息化环境集成的尝试. 现代图书情报技术，2008，24(2)：14-18.

[3] 郑刚，彭宏，郑启伦. 存储过程在嵌入式多功能数据挖掘器中的应用. 计算机应用，2010，26(6)：102-104.

[4] 徐晓霞，崔荣一，洪炳镕. 基于.NET的科研管理系统实现. 哈尔滨工业大学学报，2006，38(2)：301-303.

[5] 马洪江，周相兵. 基于存储过程的科研绩效统计分析系统的实现. 计算机技术与发展， 2011，21(8)：181-184.

[6] 吴天德，周相兵. 利用开源软件设计实现科研统计分析系统. 科技管理研究，2012，32(1)：162-166.